PHOTO RESEARCH®

Toolkit Operating Manual for the PR-704/714

Software Version 1.0

MP MASTER
MA30228

PHOTO RESEARCH

Toolkit Operating Manual for the PR-704/714

Software Version 1.0

March, 1993

Manual Release Number: 9903-0006-23 Rev. B

PHOTO RESEARCH
The Light Measurement People®

Division of Kollmorgen Instruments Corporation
9330 De Soto Avenue, P.O. Box 2192
Chatsworth, CA 91313-2192
Phone: (818) 341-5151
Fax: (818) 341-7070
TLX: 69-1427
Cable: SPECTRA

NOTICE:

The information contained in this document is subject to change without notice. If there are any questions pertaining to the use of this instrument, contact Photo Research.

Photo Research shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, peformance, or use of this material or instrument.

This document contains proprietary information which is protected by copyright. All rights are reserved worldwide. No part of this document may be photocopied or reproduced without the prior written consent of Photo Research. Patents pending.

Copyright © 1990 by Photo Research

All patents and trademarks are property of Kollmorgen Corp.

PR is a registered trademark of Photo Research.

Photo Research is a registered trademark of Photo Research.

Printed in the U. S. A.

Customer Support

Photo Research has support services available to users in case of problems using Toolkit. Photo Research personnel are available from 8:45 a.m. to 3:45p.m. (Pacific time), Monday through Friday, except on legal holidays to aid in troubleshooting system problems. The telephone number is (818) 341-5151.Before calling the Photo Research Application Engineers, please have the following information ready:

- Instrument serial number
- Instruction Manual
- Hardcopy printouts, where available, to be faxed to Photo Research to aid in troubleshooting. The fax number is (818) 341-7070

Table of Contents

Chapter 1
What Is Toolkit?

Introduction 1–1

Chapter 2
Installing Toolkit

Overview 2–1

The INSTALL Program 2–2

Chapter 3
Loading Toolkit

Overview 3–1

Loading Toolkit 3–2

Chapter 4
Using Toolkit

Overview 4–1

Sample Command Lines 4–11

Making Measurements Using UCICL Commands 4–14

Listing 1 - Sample Toolkit Measurement Program 4–16

Chapter 5
Toolkit Virtual Disk Files

Overview 5–1

Directory of Virtual Disk 5–2

Accessing Data Files 5–14

Chapter 6
Pointers

Overview 6–1

Detail of Toolkit Pointers 6–2

Loading Pointers 6–7

Variables That Control Sensitivity Parameters 6–9

Variables That Control Timing Parameters 6–11

Raw And Calculated Data Access Using Pointers 6–13

Calculation Routines 6–15

Appendices

Appendix A
Error Codes

Overview A–1

Parser Errors A–2

Measurement Errors A–5

Appendix B
Sample Source Code Listings

Overview B–1

Listing 1 - PTRDEM.C B–2

Listing 2 - SPEED.C B–19

Listing 3 - TOOLTST1.C B–34

Listing 4 - TOOLTST2.C B–38

Appendix C
TOOLSTRU

Overview C–1

Listing 1 - TOOLSTRU C-2

Listing 2 - PR704STR.H C–38

Chapter 1
What Is Toolkit?

Introduction

The **PR-700/PC Toolkit** is a memory resident program designed to provide a flexible control interface to the **PR-704** or **PR-714 SpectraScan Systems**. In operation, Toolkit becomes an extension of the **MS DOS** operating system. Two modes of operation are available.

First, the user can utilize direct measurement control of the SpectraScan directly from DOS.

Second, **Toolkit** commands may be sent by application programs written in languages that are capable of reading from or writing to files (e.g. BASIC, PASCAL, "C"). This method may include bypassing the Toolkit parser and controlling Toolkit directly through the use of **pointers**. Use of pointers is discussed in *Chapter 6.*

In general, measurements are made by copying a specific command line to the DOS reserved file **PHOTO** (a device driver), which passes the command to the resident program **TLS_RES**. The command is then parsed (decoded and checked for validity) by the command interpreter within **TLS_RES**, and the requested measurement is executed.

Following the measurement, Toolkit returns to DOS, or the next statement within an application program. The data from the measurement is retained in memory until an attempt is made to read one of the files in the **Toolkit Virtual Disk**. At this point, the data is processed and automatically sent to the appropriate file as part of the accessing process. The data on the Virtual Disk is then available in ASCII or C binary formats. Virtual Disk files are discussed in *Chapter 5.*

Minimum Hardware Requirements

- IBM PC/XT/AT 386 or compatible computer utilizing an AT I/O bus
- One, 720 K drive for non-hard disk systems
- One, 360 K floppy disk and one fixed (hard disk) for hard disk systems (recommended)
- 640 K RAM
- MS-DOS version 3.0 or later
- Math co-processor (recommended)

Software Components

Your Toolkit package consists a single, 360 K floppy disk named **PR-704** (or PR-714) **Toolkit**, and this manual. The disk contains all files necessary for the installation and operation of Toolkit. The directory of the Toolkit distribution disk and a description of the files follows.

Toolkit Disk Files

The Toolkit disk contains the following files:

```
Volume in drive A is PR704-TOOLK
 Directory of  A:\
DISK     ID        40   5-25-90  10:50a
INSTALL  DAT     8777   6-05-90  10:08a
INSTALL  EXE    80418   4-02-90   2:42p
TK704V1  LIF   254035   6-22-90   9:49a
       4 File(s)     15360 bytes free
```

DISK.ID

This file contains identification information for this disk including name of disk, version and disk title.

INSTALL.DAT

Informational parameters used by **INSTALL.EXE**.

INSTALL.EXE

The **Toolkit** installation program.

TK704VI.LIF

Compressed files that comprise the Toolkit software. This file data **must** be converted using the **INSTALL.EXE** program.

Files Created by the INSTALL Program

Following the completion of the installation, the target disk contains all files necessary for the operation of Toolkit.

The following lists **Toolkit** files as they appear after being installed to the directory **\tools** on a hard disk system. Definitions of these files follow the listing.

```
Volume in drive C is
Directory of  C:\TOOLS

.            <DIR>              1-01-80  12:02a
..           <DIR>              1-01-80  12:02a
EXAMPLES     <DIR>              1-01-80  12:04a
TLS_BLK SYS     1570  5-15-90   9:38a
TLS_CHR SYS      347  5-30-90   7:09a
TLS_RES EXE    67657  6-21-90  10:39a
DL      EXE    34483  5-28-90   9:25a
MEASINT EXE    45127  5-28-90   9:23a
CNFINT  EXE    46041  5-28-90   9:23a
CMDINT  EXE    37743  6-21-90  10:09a
STATINT EXE    19603  5-29-90  11:04a
TOOLTST EXE    12215  5-28-90   9:25a
      12 File(s)   8099840 bytes free
```

TLS_BLK.SYS

Device driver responsible for creating the Virtual (RAM) Disk that will be used by Toolkit for measurement results.

TLS_CHR.SYS

Device driver responsible for the command line communications between DOS and Toolkit. This driver and **TLS_BLK** are loaded at boot-up by commands within **CONFIG.SYS**. Your **CONFIG.SYS** file must contain the following lines for Toolkit to be executed successfully. The only changes that should be made to these commands are to the directory which contains the drivers if the default directory is not used.

```
device = c:\tools\TLS_BLK.SYS
device = c:\tools\TLS_CHR.SYS
```

TLS_RES.EXE

Main Toolkit resident program. After the aforementioned device drivers have been loaded, TLS_RES is loaded from the **Toolkit** sub-directory.

DL.EXE

Program designed to allow loading of data (**correctd.bin** and **meas.bin**) from previous Toolkit measurements into the Toolkit environment. Can be used in the absence of PR-704 or PR-714 Optical Head. Toolkit must be loaded for this program to execute successfully.

MEASINT.EXE

Creates an annotated listing of the contents of **struct meas** in an ASCII file called **MEASOUT.DAT**. This structure contains measurement parameters and corrected data. To view **MEASOUT.DAT**, run the program **MEASINT.EXE** following a measurement. Use this as a debug tool in analyzing **Toolkit** measurements.

CNFINT.EXE

Used to create an annotated version of **struct config** in an ASCII file called **CNFOUT.DAT**. This structure contains configuration and calibration information pertaining to the specific **PR-704** or **PR-714** in use.

CMDINT.EXE

Creates an ASCII file named **CMDOUT.DAT** containing the contents of **struct cmd** with descriptions. Also holds contents of **struct cmd_wndw** and **struct adapt**. **Struct cmd_wndw** contains information regrading measurement set-up parameters, and **struct adapt** sets the Adaptive Mode hardware parameters.

STATINT.EXE

STATINT.EXE reports the contents of **struct hd_stat** as described in **PR704STR.H**, and creates the file **STATOUT.DAT**. The user should be aware that a number of the flags contained in this structure are for **spatial** measurements, and are **not applicable** for the **PR-704** and **PR-714** systems.

TOOLTST.EXE

An executable program that performs a simple measurement by sending the command **X*** to **PHOTO**. Following the measurement, the program exits back to DOS and calculated data files are available on the Virtual (RAM) Disk.

Toolkit Program Examples

The following directory lists files available on the sub directory **\tools\examples** as they would appear installed on a hard disk system. Explanations of these files follow the listing.

Volume in drive C is

Directory of C:\TOOLS\EXAMPLES

```
.            <DIR>              1-01-80  12:04a
..           <DIR>              1-01-80  12:04a
PTRDEM           231   5-08-90  10:25a
PTRDEM   C     16729   5-16-90   4:40a
SPEED            224   5-08-90  10:12a
SPEED    C     13095   5-16-90   4:59a
TLSSTR4  H     44028   5-14-90  10:19a
PR704STR H      8302   4-09-90   3:19p
TOOLTST1          60   1-05-90   4:08p
TOOLTST1 C      2979   5-16-90   5:06a
TOOLTST2          60   1-05-90   4:08p
TOOLTST2 C      3419   5-16-90   5:09a
      12 File(s)  8099840 bytes free
```

PTRDEM and PTRDEM.C

PTRDEM is the make file for the source code **PTRDEM.C**. **PTRDEM.C** is a "C" source code example for using pointers to control **Toolkit**. This and all source code examples have been compiled and tested using the **Microsoft C Compiler version 5.1.** Listings of all sample program source codes can be found in **APPENDIX B.**

SPEED and SPEED.C

SPEED is the make file for **SPEED.C. SPEED.C** is a sample code demonstrating a series of measurements that are used to determine speed comparisons.

TLSSTR4.H

This header file defines **TOOLSTRU**, the main template containing the structures used within **Toolkit.**

PR704STR.H

Header file that defines **struct hd_stat**, used for Optical Head parameters.

TOOLTST1 and TOOLTST1.C

TOOLTST1 is the make file for **TOOLTST1.C. TOOLTST1.C** is a sample program that uses pointers to make a single measurement and retrieve the calculated data.

TOOLTST2 and TOOLTST2.C

TOOLTST2 is the make file for **TOOLTST2.C**. **TOOLTST2.C** is a sample program that makes a single measurement. It differs from **TOOLTST1.C** in that it uses the conventional method of writing a command to the reserved file **PHOTO** instead of using pointers to initiate the measurement.

Chapter 2
Installing Toolkit

Overview

Toolkit *must* be installed to another storage medium before it can be used. This is accomplished using the **INSTALL** program.

Toolkit may be installed on a **720 K** (or larger) floppy drive, or **hard drive (recommended). It cannot be installed on a 360 K drive.**

The 720 K drive must be used as the booting disk for the system, and **must** be formatted with **DOS** system files (**format /s**). This will insure proper loading of **Toolkit** drivers when the computer is booted up.

In addition, this disk **must** contain the calibration files for the **PR-700** that are included in **SpectraScan** or **SpectraView** software (after installation, not from the distribution disk). These files begin with the serial number of the instrument (e.g. **1005ap1.cal**) and are typically found in the sub-directory **\spectra\cal** (default). See the **SpectraScan** or **SpectraView** Instruction Manual for full details.

The INSTALL Program

The **INSTALL** program (**INSTALL.EXE**) is an executable program which leads the user through the procedure with a minimum of interaction. It automatically determines the hardware configuration of your PC, creates appropriate sub-directories on the target drive and transfers and decompresses all necessary files from the distribution (**Toolkit**) disk.

Installation Procedure

The following instructions will guide the user in the successful installation of **Toolkit** on a hard or floppy disk system.

Step 1 Insert the **Toolkit** disk in a floppy disk drive capable of reading **360K** floppy disks (e.g. **A>** or **B>** drive)

Step 2 Type the letter of the drive containing the **Toolkit** disk, followed by a colon, then press **Enter**
example - A: or B:

Step 3 Type **INSTALL**, then press **Enter**

The first screen encountered is a **system hardware configuration** screen that is a result of polling by the **INSTALL** program

Step 4 Press any key to continue, or **Esc** to quit

The second screen also displays system information.

Step 5 Press any key to continue, or **Esc** to quit

The next screen is the target drive selection screen. A listing of available drives appears.

Step 6 Use the **UP/DOWN** arrow keys to highlight the desired drive, then press **Enter**

If a hard disk is installed, the highlight defaults to that drive letter (**e.g. C:**). If the drive is a floppy drive, the disk (**720K** minimum) must be **formatted**, and contain **DOS system files**, the **COMMAND.COM** file and calibration files from an installed version of **SpectraScan** or **SpectraView** software.

Next, the directory (disk path) screen is displayed. The information selected in this screen will inform the **INSTALL** program of the directory on the destination drive where Toolkit is to be installed. The default directory is **\tools**.

Step 7 If the drive already has a directory named **\tools**, change the default by using the **right/left arrow keys**

Step 8 Enter the **new sub-directory**

Step 9 To **accept** the currently **displayed directory**, press **Enter**

The INSTALL program will proceed to copy and decompress files to the target drive.

The last task of the **INSTALL** program is to add a file named **CONFIG.NEW** to the root directory of the destination drive. The commands in this file are:

```
(hard disk example)
DEVICE=C:\TOOLS\TLS_CHR.SYS
DEVICE=C:\TOOLS\TLS_BLK.SYS
FILES=20
```

and **must** be appended to, or used to create a file named **CONFIG.SYS** on the root directory of the system booting disk. Note the disk path specification instructing **CONFIG.SYS** where to find the device drivers.

This is accomplished in one of two fashions:

1 - If a **config.sys file already exists** on the root directory of the booting disk:

Type **copy config.sys + config.new config.sys,** then press **Enter.**

This appends config.new to config.sys.

2 - If there is **no CONFIG.SYS** file on the root directory of the booting disk:

Type **copy config.new config.sys,** then press **Enter.**

This creates a new config.sys file with instructions from config.new.

NOTE:
The Toolkit program will not run without modifying the CONFIG.SYS file per the preceding procedures.

Toolkit is now installed. To load **Toolkit**, proceed to *Chapter 3.*

Chapter 3
Loading Toolkit

Overview

Now that the basic elements of **Toolkit** have been installed, the user is ready to load **Toolkit**. For first time execution, or for installation on a new computer (PC), this includes tailoring **Toolkit** to your PC configuration. This is accomplished by running the program **TLS_RES** (also used for subsequent execution of **Toolkit**), and is explained below.

Loading Toolkit

NOTE:
To successfully load Toolkit, the PR-700/PC Interface Board must be installed in the PC, and the PR-704/714 Optical Head connected. For complete details, consult the SpectraScan or SpectraView Instruction Manual.

Step 1 Following installation from the distribution disk, the computer must be re-booted (press **Ctrl Alt Del** or turn the computer off and then on)

This will cause the device drivers specified in **CONFIG.SYS** to be executed. If the computer has been re-booted prior to reaching this step, skip to **Step 2.**

The message

```
TOOLS installing virtual disk D:
        Buffer size:  48KB
        Sector size:  128
        Directory entries:  64
```

is displayed indicating the device drivers have been loaded successfully, and the RAM disk assigned by Toolkit is, in this case, labeled drive **D:**

Loading TLS_RES

Step 2 Type **TLS_RES**, then press **Enter**

If you are loading **TLS_RES** for the first time, the following message is displayed:

```
TLS_RES PR-704 Vers. 1.00
Copyright (C) Photo Research
1986-1990. All rights reserved
```

If this is not the initial execution of **TLS_RES**, skip to section *Loading TLS_RES (cont.)*.

NOTE:
To change parameters described below, delete the file TOOLS.CFG from the Toolkit directory before loading TLS_RES.

Building the TOOLS.CFG File

Immediately following the copyright screen, the user is instructed to enter configuration information in order to build the file **TOOLS.CFG**. Five parameters will need to be specified and include:

(1) **pathname for configuration files**

(2) **i/o base address**

(3) **serial number of PR-704 (or 714)**

(4) **auxiliary file name**

(5) **interrupt base vector**

These parameters are explained below and must be entered **if** the following screen appears.

```
Can't find TOOLS.CFG, Please enter the following information
Input pathname for config files
[\spectra\cal ]...
```

(1)

The config files referred to are calibration files found in **SpectraScan** or **SpectraView** software for the **PR-704** (or **714**). The default directory (**\spectra\cal**) is the directory created when installing **SpectraScan** or **SpectraView**. The calibration filenames start with the serial number (e.g. **1001ap1.cal**).

Press **Enter** to accept the default, or **input a new pathname** then press **Enter**.

```
Input io_base address [608]...
```

(2)

This value, in decimal, is the PC bus base **input/output address.** The factory default is **608** decimal (**260H**).

Press **Enter** to accept the current value, or change to a value that corresponds to the current setting on the **PR-700/PC Interface Board.**

Consult the **SpectraScan** or **SpectraView Instruction Manual** under **Getting Started** for complete information on changing the **I/O Address** on the **PR-700/PC** board.

(3) Input head serial number [0000]...

Type the **PR-704 serial number** (e.g. 1001), then press **Enter**.

(4) Input filename for auxiliary file [sub.fil]...

This file is used by **Toolkit** to temporarily store information. The only time it should be changed is if there is an existing file named sub.fil in the **Toolkit** directory.

Press **Enter** to accept the **default.**

(5) Input Interrupt Base Vector [0X60]...

Base interrupt used by Toolkit. Range is **60H - 67H**. Change only if the default value conflicts with existing software.

Press **Enter** to accept **default.**

Loading TLS_RES (cont.)

TLS_RES will now load calibration files from the specified **config** directory and perform a hardware status test. This is followed by the message:

TOOLKIT LOADED OK

Toolkit is now ready for use.

Chapter 4
Using Toolkit

Overview

Toolkit can be used to make measurements with the **PR-704/714** in two ways: using **command lines**, and with **pointers**.

Command lines can be executed directly from **DOS**, or from an **application program**.

Pointers are utilized strictly with a "C" program.

This chapter deals with Toolkit commands, and the procedures for sending commands directly from **DOS** and from within an application program. Use of **pointers** is discussed in *Chapter 6*.

When making measurements directly from **DOS**, or from within an application program, the same basic technique is utilized. The technique consists of sending a command line comprised of specific syntax to the **Toolkit** reserved filename **PHOTO**. The command line is then parsed (decoded) by the character driver, and the requested measurement is performed. Following the measurement, data collected from the measurement can be accessed from files on the **Virtual Disk** created by Toolkit during the boot-up of the PC. These files are discussed in *Chapter 5*.

UCICL Commands

UCICL (Universal Computer Interface Command Language) commands consist of reserved phrases specific to **Toolkit.** These phrases are divided into six major categories:

(1) **mode clauses**

(2) **window clauses**

(3) **set-up clauses**

(4) **trigger clauses**

(5) **calculation clauses**

(6) **execute clause**

(1) Mode Clause

Format - wav;

The **wav** mode clause specifies a wavelength based on spectral measurement. The mode clause must be included whenever **window**, **set-up**, **calculation** or **trigger** clauses are specified in the command line.

(2) Window Clauses

Format - Sttt,Euuu,Mv,Gw.www,Lx

Window clauses are optional parameters that can be included in a UCICL command to specify instrument set-up parameters. Once executed, the values defined in a window clause become the default until changed by another window clause. The Toolkit window clauses and their meanings follow.

Sttt - (**ttt** is Integer) Starting wavelength in nanometers. The value **380** must always be used for **PR-704** or **PR-714** systems.

Euuu - (**uuu** is Integer) Ending wavelength in nanometers. Possible values for **uuu** are:

1 - 780 for **PR-704** systems.

2 - 1070 for **PR-714** systems.

NOTE:
The S and E (Starting and Ending Wavelengths) must be included in all window clauses. All other window clause parameters are optional. If no window clause parameters are to be defined, the entire window clause may be omitted. See Sample Command Lines.

[Mv] - (v is Integer) Number of cycles to average. Range of values for v is **1** (**power-up default**) to **99**.

[Gw.www]- (**w.www** is Float) Measurement integration time. Select adaptive integration by setting **w.www** to **0.0** (Power-up default). Range for fixed integration times is **0.026** (26 milliseconds) to **60.0** seconds.

[Lx] - (**x** is integer) Specifies a Light and/or Dark measurement to be performed during a measurement cycle.

Possible values for **x** are:

0 - Light only. No Dark (current) measurement performed.

1 - Dark (current) only. No Light measurement performed.

2 - Both Light and Dark. Power up default.

(3) Set-Up Clause

The set-up clause specifies which optical accessory(s) and aperture will be used during a measurement. By specifying an accessory or aperture, Toolkit is instructed to use the proper calibration factors in the calculation of corrected data.

Format - [Uwww,xxx,xxx,xxx,yyy]

Possible values for **www** and **xxx** can be found by using the program **ACCINT.EXE** which is installed in the Toolkit directory (default is **\tools**). **ACCINT.EXE** can only be run after **TLS_RES** has been successfully loaded.

www - **www** (integer) values are found in the **Code** column of the **ACCINT.EXE** output display that correspond to a **Lens Flag** value of **1** (primary accessory). A primary accessory is an accessory that **replaces** the Standard Objective Lens (usually the MS-55). There may only be **one www** value entered in a setup clause. The exception to this rule is the **Standard Objective Lens**. When specifying an **add-on accessory** for use with the Standard Objective Lens, the **xxx** value is omitted, and the **yyy** variable (aperture) **must** be specfied.

xxx - xxx (integer) values are also found in the **Code** column of the **ACCINT.EXE** output display and correspond to a **Lens Flag** value of **0** (add-on accessory). An add-on accessory is one that can be used in combnation with a primary accessory (example: ND-700-10 neutral density filter). A total of three add-on accessories may be specified.

Possible values for yyy are: (**Not applicable for single aperture systems**)

301 - (Integer) For dual (**/DA** option) aperture systems, selects Aperture #1 (Aperture A).

302 -(Integer) Selects Aperture #2 (Aperture B). **Power-up default.**

A sample of the **ACCINT.EXE** output screen follows some Set-Up clause command line examples:

Example 1 - WAV;U10;X;*

Command Toolkit to use primary accessory with a code of 10 (MS-77 from following screen sample).

Example 2 - WAV;U14,301;X;*

Use add-on accessory with a code of 14 (RSC-1 from following screen) with the Standard Objective Lens, and the aperture A.

Example 3 - WAV;U10,11;X;*

Select primary accessory 10 for use with add-on accessory 11 with the current aperture.

(4) Calculation Clause

The calculation clause serves two purposes. It allows the user to specify **CIE 2^0 or 10^0 Standard Observer** data for photometric calculations, and selection of **power** or **energy** units.

Format - [Lax or Lqy]

Lax - (Integer) Select CIE data set for photometric calculations. Possible values for **x** are:

2 - Select **CIE 2^0** data. **Power-up default.**

10 - Select **CIE 10^0** data.

Lqy - (Integer) Select "**power**" or "**energy**" units. Possible values for **y** are:

0 - **Power units.** Used when measuring continuous (non-pulsed) sources. **Power-up default.**

1 - **Energy units.** Results are in units/sec (e.g. footLambert seconds). Used for pulsed sources (e.g. strobe light). Pulsed light measurements require optional hardware. Consult factory.

To calculate DC (continuous) values from energy units, use the following formula:

$$X_{dc} \quad \frac{X_e}{E_n \times E_t}$$

where:

X_{dc} = DC equivalent (e.g. footLamberts)

X_e = Energy units (e.g. footLambert seconds)

E_n = Number of events

E_t = Elapsed time of single event

(5) Trigger Clause

The Trigger clause allows you to tell Toolkit to wait for a keystroke before starting a measurement, or synchronize the start time of the measurement with an external event. Both of these trigger clauses may be used in the same command line.

If ordered, two mechanisms are provided for external synchronization purposes: **SYNCH IN**, and **SYNCH OUT** (Consult factory for optional Pulsed Light Hardware).

SYNCH OUT - Pulses are sent to the SYNCH OUT connector on the cover plate every time a measurement cycle is initiated from the SYNC IN port.

SYNCH IN - Signals can be used to initiate measurements, by sending a TTL level signal to the SYNCH IN connector on the same cover plate.

Format - [Tx or Tm]

Tx - Allows the user to initiate a measurement by waiting for a signal to the **SYNCH IN** connector. **x** is sent over as a literal character and does not represent a variable.

NOTE:
When using the Tx Trigger clause, the window clause G must be set to a fixed integration time that is longer than the event. Use of Adaptive Sensitivity will yield unexpected and erroneous results.

Tm - Commands Toolkit to wait for a keystroke on the PC before beginning the measurement. **m** is sent over as a literal character and does not represent a variable.

(6) Execute Clause

Format - X*

To make a measurement using the most recently defined set-up parameters (integration time, accessory, aperture etc.), simply send **X*** to the reserved device driver **PHOTO**. For expanded command lines, the **execute clause** must be placed at the end of the line. Once the parser encounters a "*", it assumes the command line is complete and attempts to execute the measurement.

Sample Command Lines

Following are sample **Toolkit (UCICL)** command lines and their descriptions. Any of these command lines may be sent directly from DOS, or from within an application program.

The command line descriptions appearing on the following page(s) use the following conventions:

1 - Upper case single letters and mnemonics are reserved phrases that denote their function.

2 - Lower case letters indicate variable parameters such as starting wavelength.

3 - [...] indicates optional specifications(s), but is not actually used in the command line.

4 - Commands are composed of clauses.

5 - Each clause specifies one aspect of the command.

6 - Clauses are separated by semicolons (;).

7 - Parameters within a clause are separated by commas (,). Parameters within a window clause may be omitted if not required.

8 - Command lines are terminated by a star (*) character.

9 - The clauses in a command must be defined in the order shown in the command description below. However, entire clauses may be omitted if not required for the specific measurement.

10 - Blanks, tabs, carriage returns and line feed characters may be inserted anywhere in command lines to aid readability.

UCICL command lines have the following format:

[mode clause] [; window clause,window clause...] [; trigger clause] [; set up clause] [; calculation clause;] execute clause *

Example 1 - X* or [WAV;]X*

Command **Toolkit** to initiate a measurement based on most recent set-up parameters as defined by window clause, set-up clause etc. If sent directly after **TLS_RES** is loaded, **power-up default** settings are used. Both variations perform the same function, therefore **WAV;** is optional.

Example 2 - WAV;U2,301;X*

Perform a measurement using primary accessory number 2 ,(**U2**) and aperture number 1 (**301 [for dual aperture systems only]**). Note the exclusion of window clauses as they are optional if selecting apertures or accessories only. **WAV;** (mode clause) is **required** because of the set-up clause.

Example 3 - WAV;S380,E780,M3,G1.50;X*

Specify a measurement that averages three measurement cycles and fixes the integration time to 1.50 seconds. Note the **S** and **E** window clause parameters **must** be included if subsequent window clauses are specified. Also, as in Example 2, the mode clause is required.

Example 4 - WAV;S380,E780,G1.0;Lq1;Tx;X*

External trigger measurement (consult factory regarding optional Pulsed Light Hardware). Fixed integration time must be specified when externally triggering a source. In this example, the integration time is set to 1.0 seconds as defined by **G1.0**. External trigger clause (**Tx**) waits for a signal from the **SYNC IN** port, and upon receiving, sends a TTL level signal to the **SYNC OUT** port (SYNC IN and SYNC OUT are optional - Consult Factory). **Lq1** specifies temporal (time) based corrected values (e.g. footLambert seconds). Window clause parameters **S380** and **E780** **must** be included because of **G1.0** window clause.

Example 5 - WAV;La10;X*

Request photometric values be calculated using CIE **10^{o} Standard Observer** Data (**La10**). Note absence of window clause demonstrating optional nature. If no new window parameters are included, then parameters last defined in window clause (or default if no window clauses have been defined since loading TLS_RES) are used.

Making Measurements Using UCICL Commands

From DOS

To make measurements directly from DOS, two options are available: from an **existing file**, or **directly from the keyboard.**

From An Existing File

Step 1 Create a file with the desired command line

In this example, we will use the execute clause (**X***). The file can be created using a text editor, or with the DOS **copy con** utility. To create a file using **copy con**:

Step 1A From the DOS prompt, type **copy con filename**, then press **Enter** (**filename** can be any legal DOS filename)

Step 1B Type **X*** (**X** may be lower case), then press **Enter**

Step 1C Press **Ctrl** and **Z** simultaneously (or press the **F6** function key), then press **Enter**

The message 1 File(s) copied should appear on-screen.

Step 2 Type **copy filename photo**, then press **Enter**

At the prompt 1 File(s) copied, the measurement is complete.

Directly From the Keyboard

Step 1 Type **copy con photo**, then press **Enter**

Step 2 **Type X*** (UCICL command line), then press **Enter**

Step 3 Press **Ctrl and Z** (or **F6**), then press **Enter**

At the prompt 1 File(s) copied, the measurement is complete.

From An Application Program

Step 1 Open the file **PHOTO** for sequential output

Step 2 Write the **UCICL** command line to **PHOTO**

Once Toolkit returns control to the program, the measurement is complete

Listing 1 - Sample Toolkit Measurement Program

The following example is coded in interpretive BASIC, and demonstrates the preceding steps.

```
10 REM        Program to make a Toolkit Measurement
20 DIM WRT$[20]                    REM Dimension WRT$ - Will be used for
                                   REM UCICL command line
30 WRT$ = SPACE$(20)               REM Initialize WRT$
40 WRT$ = "WAV;X*"                 REM Assign UCICL command line to WRT$
50 OPEN "O",#1,PHOTO               REM Open PHOTO for sequential write as  REM #1
60 PRINT #1,WRT$                   REM Send UCICL command to PHOTO
70 CLOSE #1                        REM Close file buffer #1
80 PRINT "Back From Measurement"   REM Indicate return to
                                   REM program control
90 END
```

Chapter 5
Toolkit Virtual Disk Files

Overview

At the conclusion of a measurement, data from the measurement is made available as files on the **Virtual** (RAM) **Disk** as created by **Toolkit**. The **Virtual Disk** is designated as the next logical drive following physical drives. For example, for most hard disk systems, the **Virtual Disk** is designated as drive **D:**.

Virtual Disk files can be accessed as ASCII text or "C" binary files. All files correspond, item by item, to their appropriate structure located in **TOOLSTRU** (stored as **TLSSTR4.H** in the **/EXAMPLES** directory) and documented in **APPENDIX C**. The first item of each file will be the number of fields if it is an **.ASC** file and number of **bytes** if it is a **.BIN** file.

ASCII files provide an easy means of accessing data through **DOS**, or higher level programming languages. All **ASCII** files are denoted by the file extension **.ASC**. Binary files have the advantage, when used by C programs, of speed and compactness and have the extension **.BIN**.

NOTE:
All files on the Virtual Disk are to be regarded as read-only files. Any attempt to write to these files will result in unexpected memory re-allocation and may cause a system failure.

Directory of Virtual Disk

```
 Volume in drive D is TOOLS  V1.0
 Directory of  D:\
POINTERS BIN     236  7-20-90  12:28p
CIE      ASC     258  7-20-90  12:28p
CIE      BIN      60  7-20-90  12:28p
CORRECTD ASC    3460  7-20-90  12:28p
CORRECTD BIN     896  7-20-90  12:28p
RAWDARK  ASC    2225  7-20-90  12:28p
RAWDARK  BIN     528  7-20-90  12:28p
RAWDIFF  ASC    2225  7-20-90  12:28p
RAWDIFF  BIN     528  7-20-90  12:28p
RAWLIGHT ASC    2225  7-20-90  12:28p
RAWLIGHT BIN     528  7-20-90  12:28p
MEAS     ASC     600  7-20-90  12:28p
MEAS     BIN    3040  7-20-90  12:28p
CCT      ASC      54  7-20-90  12:28p
CCT      BIN      14  7-20-90  12:28p
STATUS   ASC     156  7-20-90  12:28p
STATUS   BIN      58  7-20-90  12:28p
ERRCODE  ASC      10  7-20-90  12:28p
BKGND    BIN    3040  7-20-90  12:28p
SPARE    ASC     225  7-20-90  12:28p
SPARE    BIN     222  7-20-90  12:28p
      21 File(s)    24192 bytes free
```

The directory listed above is a directory of files available on the Toolkit Virtual Disk. Following are descriptions of these files.

POINTERS.BIN

POINTERS.BIN is located in the Virtual Disk and therefore listed here, but is not a true data file. It contains **struct pointers** in binary format.

CIE.ASC (CIE.BIN)

CIE.ASC and **CIE.BIN** provides radiometric and colorimetric data based on results of the last successful measurement. The **.BIN** file is derived from **struct calc_w** (See **APPENDIX C**). A typical **CIE.ASC** file follows.

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000017	int	number of fields in this array
2	000000	int	N/A
3	000001	int	N/A
7	8.837524e+001	float	1931 CIE X value
8	8.934883e+001	float	1931 CIE Y value (in **cd * m^{-1}** or **lux**)
9	5.437158e+001	float	1931 CIE Z value
10	0.000000e+000	float	1931 CIE V (Scotopic luminance in **cd * m^{-2}** if calculated luminance is below 0.3 cd * m^{-2})
11	2.607771e+001	float	1931 CIE Y converted to English units (**footLamberts** or **footcandles**)
12	0.000000e+000	float	Scotopic luminance in footLamberts
13	3.807707e-001	float	1931 CIE x
14	3.849656e-001	float	1931 CIE y
15	2.220870e-001	float	1960 CIE u
16	3.368005e-001	float	1960 CIE v
17	5.052008e-001	float	1976 CIE v' 1960 CIE u = 1976 CIE u')

CORRECTD.ASC (CORRECTD.BIN)

CORRECTD.ASC contains corrected spectral radiance (or irradiance) from the last measurement. This is the one set of calculations performed following each measurement, whether requested or not.

For **PR-704** systems, the data is reported every **2 nm**, for the **PR-714** the data is at **4 nm** intervals. The following sample **CORRECTD.ASC** file and its descriptions are derived form **struct meas_data**. For **CORRECTD.BIN** see **APPENDIX C**.

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000226	int	Number of fields
2	000001	int	N/A
3	000201	int	Number of data points
4	000000	int	Spare
5	000000	int	Spare
6	000000	int	Spare
7	5.850184e-005	float	First corrected data point (380 nm)
.			
.			
.			
207	4.985446e-005	float	Last corrected data point (780 nm for PR-704, 1070 nm for PR-714)
208	0.000000e+000	float	Spare
209	0.000000e+000	float	"
210	0.000000e+000	float	"
211	0.000000e+000	float	"
212	0.000000e+000	float	"
213	0.000000e+000	float	"
214	0.000000e+000	float	"
215	0.000000e+000	float	"
216	0.000000e+000	float	"
217	0.000000e+000	float	"
218	0.000000e+000	float	"
219	0.000000e+000	float	"
220	0.000000e+000	float	"
221	0.000000e+000	float	"
222	0.000000e+000	float	"
223	0.000000e+000	float	"
224	0.000000e+000	float	"
225	0.000000e+000	float	"
226	0.000000e+000	float	"

RAWDARK.ASC (RAWDARK.BIN)

This file contains uncorrected detector data as the result of the last **Dark** (current) measurement. It is derived from **struct meas_data**. To view this structure for binary file applications, run the program **MEASINT.EXE** following a measurement, and view the file named **MEASOUT.DAT** (ASCII) under **Start of raw dark data.**

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000262	int	Number of fields
2	000001	int	Number of dimensions
3	000256	int	Dimension size #0 (256 detectors)
4	000000	int	Dimension size #1
5	000000	int	Dimension size #2
6	000000	int	Dimension size #3
7	000571	int	Raw dark data number 000 (detector #1)
	.		
	.		
	.		
262	000106	int	Raw dark data number 255 (detector #256)

RAWDIFF.ASC (RAWDIFF.BIN)

This file contains uncorrected detector data as the result of the last (Light - Dark) measurement. It is derived from **struct meas_data**. To view this structure for binary file applications, run the program **MEASINT.EXE** following a measurement, and view the file named **MEASOUT.DAT** (ASCII) under **Start of raw light minus dark data.**

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000262	int	Number of fields
2	000001	int	Number of dimensions
3	000256	int	Dimension size 0 (256 detectors)
4	000000	int	Dimension size 1
5	000000	int	Dimension size 2
6	000000	int	Dimension size 3
7	000001	int	Light - Dark data point 0 (detector #1)
	.		
	.		
	.		
262	000007	int	Light - Dark data point 255 (detector #256)

RAWLIGHT.ASC (RAWLIGHT.BIN)

This file contains uncorrected detector data as the result of the last **Light** measurement. It is derived from **struct meas_data**. To view this structure for binary file applications, run the program **MEASINT.EXE** following a measurement, and view the file named **MEASOUT.DAT** (ASCII) under **Start of light data.**

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000262	int	Number of fields
2	000001	int	Number of dimensions
3	000256	int	Dimension size 0 (256 detectors)
4	000000	int	Dimension size 1
5	000000	int	Dimension size 2
6	000000	int	Dimension size 3
7	000572	int	Raw light data point #0 (detector #1)
	.		
	.		
	.		
262	000113	int	Raw light data point #255 (detector #256)

MEAS.ASC (MEAS.BIN)

MEAS.ASC contain a number of hardware and calculation parameters determined during the most recent measurement. It is derived from different elements of **struct meas** including **struct cmd_wndw wn[0]**, **struct_adapt** and **struct result_wndw wn[0]**. **MEAS.BIN** is the entire **struct meas** and contains all parameters and data contained in that structure. For full details of **struct meas**, view the file **MEASOUT.DAT** that is created by the executable program **MEASINT.EXE**, or see **APPENDIX C.**

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000040	int	Number of fields (from struct meas)
2	005900	int	Toolstru version date. In this case, 05_90. (From struct meas)
3	???***$PhotoR$***???		idate (from struct meas). This string is dummy string.
4		char	Measurement description (from struct meas)
5	07/20/90	char	Measurement date (from struct meas)
6	13:24:48	char	Measurement time (from struct meas)
7	0000000000 long		Time of measurement completion in milliseconds since GMT 00:00:00 1/1/70 (from struct meas)
8		char	Serial number of PR-700 (from struct meas)
9	000000	int	Measurement ID number (from struct meas)
10	000001	int	Number of measurement windows. Always 1. (from struct cmd_wndw)
11	3.800000e+002	float	Starting wavelength (from struct cmd_wndw)
12	7.800000e+002	float	Ending wavelength (from struct cmd_wndw)
13	2.000000e+000	float	Wavelength interval for corrected data (from struct cmd_wndw)

FIELD	CONTENTS	TYPE	DESCRIPTION (cont.)
14	5.000000e+000	float	Spectral bandwidth. Varies with different apertures. (from struct cmd_wndw)
15	9.999000e+003	float	Integration time. 9.999e+003 indicates Adaptive Sensitivity. (from struct cmd_wndw)
16	000004	int	Detector type. Always 4 for 704/714. (from struct cmd_wndw)
17	000001	int	Number of measurements averaged (from struct cmd_wndw)
18	000000	int	Measurement type. 0 = Power, 1 = Energy (from struct cmd_wndw)
19	000002	int	Light/Dark flag. 0 = Light only, 1 = Dark only, 2 = Light and Dark (from struct cmd_wndw)
20	000000	int	Measuring shutter flag. 0 = closed during dark measurement, 1 = shutter open (from struct cmd_wndw)
21	000000	int	Data quality. (from struct result_wndw)
22	000000	int	Low range indicator. Not used in PR-700 systems. (from struct result_wndw)
23	000000	int	Range indicator. N/A for PR-700. (from struct result_wndw)
24	000000	int	Run flag. Always 0. (from struct result_wndw)
25	000000	int	Calculation flag. Always 0. (from struct result_wndw)
26	000001	int	Measurement flag. 1 = Successful measurement complete (from struct result_wndw)
27	000001	int	Corrected data available flag. 1 = true (from struct result_wndw)
28	000001	int	Cie calculated flag. 1 = true (from struct result_wndw)
29	000001	int	CCT calculated flag. 1 = true (from struct result_wndw)
30	000000	int	Flying height calculated flag. 1 = true. (from struct result_wndw). Not used for 704/714.
31	000000	int	Spatial flag. Always false for PR-704/714. (from struct result_wndw)

FIELD	CONTENTS	TYPE	DESCRIPTION (cont.)
32	000000	int	we_them flag. 1 = factory measurement, 0 = customer meas. (from struct result_wndw)
33	000000	int	Number of accessories used for this meas. Max is 4. (from struct result_wndw)
34		char	Accessory #1 description
35		char	Accessory #2 description
36		char	Accessory #3 description
37		char	Accessory #4 description
38	000000	int	Radiance flag. 0 = radiance, 1 = irradiance, 2 = radiant intensity (from struct result_wndw)

CCT.ASC (CCT.BIN)

This file contains correlated color temperature information calculated from the last measurement. For **CCT.BIN** detailed information, see **Toolstru (TLSSTR4.H or APPENDIX C)**

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000004	int	Number of fields
2	4.047158e+003	int	Correlated color temperature in Kelvins
3	2.470870e+002	int	Correlated color temperature in MK-1 (micro-reciprocal Kelvins)
4	3.710959e-003	int	Distance from Plancks locus in CIE 1960 uv Uniform Color Space

STATUS.ASC (STATUS.BIN)

STATUS.ASC contains PR-704/714 Optical Head status information from the last measurement. It is derived from **struct hd_stat** which can be found in **PR704STR.H** (in **tools\examples** directory), or **STATOUT.DAT** which is created by the program **STATINT.EXE. Struct hd_stat is not in TOOLSTRU.**

FIELD	CONTENTS	TYPE	DESCRIPTION
1	54	int	Head id
2	0	int	Controller id
3	000256	int	Number of pixels in detector
4	0000000001	long	CPU software serial number
5	0000000001	long	CPU version number
6	???***$PhotoR$***???		Firmware build date. This represents compare string replaced by date at factory
7	2	int	Number of apertures installed in Optical Head
8	0	int	View and measuring shutter types
9	000000	int	Firmware ROM checksum
10	0	int	Optional dipswitch setting on PR-704/PC I/F determining shutter to initiate special hardware conditions and diagnostics at power-up. User setting always 0 (default).
11	0	int	Current position of measure shutter - 0 = open, 1 = closed
12	1	int	Current position of viewing shutter 0 = open, 1 = closed
13	2	int	Currently selected aperture
14	1	int	Status of thermoelectric cooler. 1 = normal
15	0	int	Status of detector housing pressure. 0 = OK
16	0	int	Current state of manual VIEW switch - 0 = not pressed, 1 = pressed

FIELD	CONTENTS	TYPE	DESCRIPTION (cont.)
17	0	int	Current state of manual measuring shutter switch. Not used.
18	000062	int	Number of faults pending
19	0	int	Current shutter mode. 0 = AUTO, 1 = MANUAL
20	0	int	Current sync mode (1 = external)
21	000010	int	Current 10X pixel clock rate in microseconds. 10 is default
22	000026	int	Minimum integration time

ERRCODE.ASC (ERRCODE.BIN)

This file holds a single integer which is the result of error codes generated by the last measurement including parser or instrument errors. For complete details on error codes, see *Appendix A*.

BKRND.BIN

BKRND.BIN is a reserve file designed for systems (such as flying height systems) that require two measurement files to compare as part of a normal measurement cycle. It is an exact duplicate of **struct meas (MEAS.BIN)**.

SPARE.ASC (SPARE.BIN)

General purpose file designed to hold auxiliary calculations. Derived from **struct spare** that can be found in *Appendix C*.

FIELD	CONTENTS	TYPE	DESCRIPTION
1	000122	int	Number of fields
2-21	0.000000e+000	float	Reserved for calculated values
22-31	000000	int	"
32-41	0000000000	long	"
42-122	000000	char	"

Accessing Data Files

.ASC files are **ASCII** text files and are subject to all the rules of DOS. The files can be **typed**, **copied** and **printed** from DOS, or from within an application program.

For details on accessing data files with **pointers**, see *Chapter 6.*

Example 1 - View File on Screen

Step 1 Type **type [RAM disk:]filename**, then press **Enter**

filename is name of a .ASC file on the RAM disk.

Example 2 - Copy File to Another Disk

Step 1 Type **copy [RAM disk:] fname1 [target disk:\path\]fname2**, then press **Enter**

fname1 is the .ASC file and **fname2** is the destination filename. [] indicates optional parameters.

Example 3 - Print File

Step 1 Type **type [RAM disk:]filename prn**, then press **Enter**

This directs file output to the currently active parallel printer port.

Chapter 6
Pointers

Overview

For users with a thorough knowledge of "C" programming (especially a complete knowledge of pointers), measurement speed may be decreased by programming in "C", and including **TLSSTR4.H** (located on the sub-directory **\tools\examples**) in the program. This file contains the header **TOOLSTRU** which is a template for all structures and functions used by Toolkit.

NOTE:
It is beyond the scope of this manual to instruct the user in the "C" programming language. If the user is not familiar with "C", and in particular with pointers, it is strongly recommended that these functions be avoided.

Detail of Toolkit Pointers

Format: data (2 unsigned integers) in the form **offset:segment**

The pointers list is composed of four different groups of pointers.

The **first** group of pointers is used to access the **calculated data files** (***.ASC** and ***.BIN**) directly.

The **second** group of pointers points to the major structures in the system. Using this group enables the user to read the contents of a data structure directly, without copying to a file first as shown in **TLSSTR4.H** (**TOOLSTRU**).

The **third** group of pointers points to several calculation routines. These routines must be run prior to extracting data using group two pointers.

A **fourth** group of pointers called "hooks" are pointers used with special calculation routines. These routines are normally installed by Photo Research and require a more thorough knowledge of the system than is possible to discuss here.

All routines neither accept nor return parameters. This means that they are compatible with any language as they make no requirements on the stack. Pointers may be used by an application program by loading into the program and overlaying the appropriate **TOOLSTRU** structure. The exact expansion of the structures is detailed in **TLSSTR4.H** (See *Appendix C*).

The following list defines all pointers used by Toolkit and available to the user. It is taken directly from **TOOLSTRU (TLSSTR4.H).**

Toolkit POINTERS Listing

```
void *files;                    /* 00 - ptr to files array*/
void (*edumfunc)(void);         /* 01 - ptr to dummy function*/
void (*eacie)(void);            /* 02 - ptr to asc cie( )    */
void (*ebcie)(void);            /* 03 - ptr to bin cie( )    */
void (*eacorrct)(void);         /* 04 - ptr to asc corrected( )*/
void (*ebcorrct)(void);         /* 05 - ptr to bin corrected( )*/
void (*earawdrk)(void);         /* 06 - ptr to asc rawdark( )*/
void (*ebrawdrk)(void);         /* 07 - ptr to bin rawdark( )*/
void (*earawdif)(void);         /* 08 - ptr to asc rawdiff( )*/
void (*ebrawdif)(void);         /* 09 - ptr to bin rawdiff( )*/
void (*earawlit)(void);         /* 10 - ptr to asc rawlit( )*/
void (*ebrawlit)(void);         /* 11 - ptr to bin rawlit( )*/
void (*eameas)(void);           /* 12 - ptr to asc meas( )*/
void (*ebmeas)(void);           /* 13 - ptr to bin meas( )*/
void (*eacct)(void);            /* 14 - ptr to asc cct( )*/
void (*ebcct)(void);            /* 15 - ptr to bin ccft( )*/
void (*eastat)(void);           /* 18 - ptr to asc stat( )*/
void (*ebstat)(void);           /* 19 - ptr to bin stat( )*/
void (*eacode)(void);           /* 20 - ptr to asc eacode( )*/
void (*ebbkgnd)(void);          /* 21 - ptr to bin bkgnd( )*/
void (*easpare)(void);          /* 22 - ptr to asc spare( )*/
void (*ebspare)(void);          /* 23 - ptr to bin spare( )*/
void (*emeas)(void);            /* 24 - ptr to meas_func( )*/
struct  meas *pmm;              /* 25 - ptr to struct meas*/
```

```
struct  cmd_wndw *pwn;          /* 26 - ptr to struct cmd_wndw*/
struct  result_wndw *pr_wn;     /* 27 - ptr to struct result_wndw*/
struct  calc_w *pcalw;          /* 28 - ptr to struct calc_w*/
struct  calc_s *pcals;          /* 29 - ptr to struct calc_s*/
struct  meas_data *mmd;         /* 30 - ptr to struct meas_data*/
struct  in_data *pld;           /* 31 - ptr to struct int_data*/
struct  int_data *pdd;          /* 32 - ptr to struct int_data*/
struct  int_data *psd;          /* 33 - ptr to struct int_data*/
struct  flt_data *pcd;          /* 34 - ptr to struct flt_data*/
struct  command *pcmd;          /* 35 - ptr to struct command*/
struct  config *pcnf;           /* 36 - ptr to struct config*/
struct  status *pstatp;         /* 37 - ptr to struct status (N/A
                                   for 704/714)*/
struct  spare *pspare;          /* 38 - ptr to struct spare*/
void (*xcorrct)(void);          /* 39 - ptr to corrected calc
                                   routine*/
void (*xciecalc)(void);         /* 40 - ptr to cie calc routine*/
void (*xcctemp)(void);          /* 41 - ptr to cct calc routine*/
void (*xhook0)(void);           /* 42 - ptr to spare hook routine*/
void (*xhook1)(void);           /* 43 - ptr to spare hook routine*/
int  *hook_rtn_flag;            /* 44 - ptr to hook_rtn global
                                   used by hookx()*/
int (*zhook0)(char *auxfil);    /* 45 - ptr to hook funct @ end
                                   of init*/
int (*zhook1)(struct config *cnf,  /* 46 -ptr to hook funct @
struct command *cmd,               meas_func entry */
      struct meas *mm );

int (*zhook2)(long time,        /* 47 -  ptr to hook funct @
      int gain,                          meas_func exit */
      struct config *cnf,
      uct command *cmd,
      struct meas *mm );
```

```
int (*zhook3)(long time,                    /* 48 -ptr to hook funct @
        int gain,                               meas_dark entry */
        struct config *cnf,
        struct command *cmd,
        struct meas *mm );

int (*zhook4)(struct config *cnf,           /* 49 -ptr to hook funct @
        struct command *cmd,                    correct entry */
        struct meas *mm );

int (*zhook5)(struct config *cnf,           /* 50 -ptr to hook funct @
        struct command *cmd,                    correct exit */
        struct meas *mm );

int (*zhook6)(struct config *cnf);          /* 51 -ptr to hook funct @
                                                meas_init entry */

int (*zhook7)(struct config *cnf);          /* 52 -ptr to hook funct @
                                                meas_init exit */

int (*zhook8)(char *p,                      /* 53 -ptr to hook funct @
        struct config *ccnf,                    toolpars entry */
        struct command *ccmd,
        struct command *prot_ptr);

int (*zhook9)(char *p,                      /* 54 -ptr to hook funct @
        struct config *ccnf,                    toolpars exit */
        struct command *ccmd,
        struct command *prot_ptr );

int (*zhook10)(char *pspare);               /* 55 -ptr to hook funct @
                                            spare.asc & .bin entry */

int (*zhook11)(char *pspare,                /* 56 -ptr to hook funct @
        char *ptr1);                            spare.asc & .bin exit */

int (*zhook12)(                             /* 57 -ptr to hook funct @
        struct config *cnf,                     cctemp entry */
        struct command *cmd,
        struct meas *mm );

int (*zhook13)(                             /* 58 -ptr to hook funct @
        struct config *cnf,                     cctemp exit */
        struct command *cmd,
        struct meas *mm );
```

```
int (*zhook14)(                         /* 59 -ptr to hook funct @
        struct config *cnf,                 ciecalc entry */
        struct command *cmd,
        struct meas *mm );

int (*zhook15)(*cnf,                    /* 60 -ptr to hook funct @
        struct config *cnf,                 ciecalc exit */
        struct command *cmd,
        struct meas *mm );
};

#define NUM_PTRS    sizeof(struct pointers) / sizeof ( char *)
/*-----------------------------------------------------------------*/
```

Loading Pointers

Step 1 Load a copy of **POINTERS.BIN** and align a pointer to it of type **struct pointers**

Step 2 Make one measurement using a **UCICL** command line to initialize the system's parameters **(Optional)**

The user may use the appropriate pointers to set the aperture and accessory.

After **Step 2** is completed, subsequent measurements may be made by simply calling **meas_func( )** directly through the use of a pointer.

NOTE:
Optical Head and accessory parameters will remain as defined during the measurement in Step 2 above. They may only be changed by another UCICL command line or appropriate pointer. These and all other parameters are reset to power-up default settings each time Toolkit is loaded.

Data that can be accessed through pointers includes raw data as well as corrected and calculated parameters.

Functions that may be run through pointers include the **measurement function** and all of the **calculation functions** used to fill data files.

An example of code implementing pointers is **PTRDEM.C** (see *Appendix B)*. This program loads the pointers file, and declares a pointer of type **STRUCT POINTER** to work with it. The program then makes a large number of measurements and stores result segments in the form of raw light data in dynamically allocated free **RAM** space.

The program illustrates methods of accessing data, moving data, and running calculation functions. Users may extract code from this program to help in the construction of their own programs.

Variables That Control Sensitivity Parameters

Following are four variables accessible to the "C" programmer using pointers. These variables control the sensitivity parameters, allowing the user to change these parameters more conveniently and quickly than through the use of a UCICL command line. They are located in **struct cmd_wndw**.

intgr_speed;

Type = float.

Value is stated in **seconds**. Set this value to run a fixed measurement integration time (possible values are **.026** to **60.000** sec.), or to **ADAPT (9999 - power-up default)** for adaptive sensitivity measurements. An optimum value may be found by using the value found in **adp_last_i_time** which was set during the last adaptive sensitivity measurement. The advantage of using this constant as a fixed integration time is speed of processing. In adaptive mode, the system goes through several iterations to arrive at an ideal measurement period.

avg_method;

Type = integer.

If the value of this constant is greater than **1**, **Toolkit** will make multiple measurements and compute the average. The possible values are **1 - 99**. If the illumination is unstable, using this method improves repeatability.

NOTE:
If multiple cycles are selected for long integration times (greater than 10 seconds), it is suggested that the integration time be fixed. This will significantly shorten the overall measurement time.

l_d_flag;

Type = integer

This variable controls basic raw data measurements. Corrected data is derived from the **raw Light data minus raw Dark data**. Sometimes when multiple measurements are to be made, it is desirable to establish a raw **Dark** measurement as the reference for successive **Light** measurements. The values of this constant may be interpreted as follows:

0 = Light only
1 = Dark only
2 = Both Light and Dark. **(Power-up default)**

z_flag;

Type = integer

This variable controls the movement of the measuring shutter during the **Dark** measurement. The values of this flag may be interpreted as follows:

0 = Close shutter during Dark measurement
1 = Leave shutter open during Dark measurement

NOTE:
The z_flag parameter is currently not implemented for the 704/714 Toolkit versions. Implementation of this function is under development. If this is a necessary function to your application, please contact Photo Research for details on the release of versions supporting z_flag.

Variables That Control Timing Parameters

Following are three variables which control the timing parameters during the **adaptive sensitivity** mode. They are accessible through **struct config.**

long min_i_time;

Type = long integer

This value defines the minimum acceptable measurement period in milliseconds. If during the adaptive process, the trial measurement falls below the minimum time limit, the process will abort leaving previous parameters intact. The default value is **26** milliseconds and should be set such that **long_min_i_time > = 26** (ms).

long max_i_time;

Type = long integer

Defines the maximum acceptable measurement period in milliseconds. If during the adaptive process, the trial measurement falls above the maximum time limit, the process will abort leaving previous parameters intact. The default value is **60,000 milliseconds (60.0 seconds)** and must be set such that **max_i_time < = 60000.**

total_adp_time;

Type = long integer

This value specifies the **maximum total time** in milliseconds for all Adaptive Sensitivity tries. The default is set for **four minutes**. Sometimes during the process, it is desirable to limit the time that the system spends, and exit quickly to trap a process error.

Raw And Calculated Data Access Using Pointers

Immediately after a measurement is successfully completed, raw data is stored in the appropriate structures as found in **TOOLSTRU** (*Appendix C*). The user can read this data directly from **TOOLSTRU** without the necessity of it first being transferred to a data file.

Pointers can also be used to initiate any of the calculation routines and are used by the system when the first sector of any of the data files are read through use of a DOS command.

In each case an appropriate calculation routine is called, and executed if it has not previously been invoked since the most recent measurement. Should a user desire to circumvent the **DOS** call, pointers may be used to fill a corresponding data file.

Note that for each **.BIN** case, the **TOOLSTRU** template is accurate. The first entry is an integer which denotes the size of the file in bytes. In the case of **.ASC** files, the order of data is correct although the first entry is the number of items (fields) in the file rather than number of bytes.

The following is a list of:

1) data structures of raw and corrected data in **TOOLSTRU**

and

2) calculation routines accessible only by pointers.

Data Structures (detailed in TLSSTR4.H or *Appendix C)*

```
struct meas *pmm;                /*  pointer to struct meas*/
struct cmd_wndw *pwn;            /*  pointer to cmd_wndw which
                                     controls basic measurement
                                     criteria.*/
struct result_wndw *pr_wn;       /*  pointer to data qualifying
                                     the current measurement.*/
struct calc_w *pcalw;            /*  pointer to spectral data items.
struct meas_data *mmd;           /*  pointer to meas_data items.
                                     Note that this is a substructure
                                     of struct meas.*/
struct int_data *pld             /*  This structure holds raw light
                                     data */
struct int_data *pdd;            /*  This is raw dark data.*/
struct int_data *psd;            /*  This is light minus dark data.*/
struct flt_data *pcd;            /*  This holds corrected data.*/
struct command *pcmd;            /*  pointer to struct command*/
struct config  *pcnf;            /*  pointer to struct config*/
struct spare *pspare;            /*  pointer to struct spare which is
                                     for custom systems.*/
```

Calculation Routines

xcorrct( )

This pointer indirectly calls the **correct( )** calculation function. This calculation converts raw data to virtual 201 pixel data which is now presented in floats. This routine fills struct **meas_data**. It does not fill the corresponding **CORRCECTD.BIN** file although it does perform some of the same tasks.

xciecalc( )

The calculations performed by this routine fill **struct calc_w.**

xcctemp( )

This routine calculates the items found in **struct cctemp.**

xhook( ) and zhook( )

The hook routines are normally set up as dummy routines, and consequently do nothing. When custom software is developed, these routines may be used to provide links to application programs. When used, these routines will make it unnecessary to invade base system code, and thereby make custom programs somewhat independent to base revision levels.

Appendix A
Error Codes

Overview

Error codes are divided into two groups, **parser errors** and **measurement errors**. An error code is available in the file **ERRCODE.ASC** or **ERRCODE.BIN** in the form of a single integer on the Toolkit RAM disk.

Parser errors are created when an illegal Toolkit command is sent to the parser **PHOTO**.

Measurement errors occur when a condition exists preventing the successful conclusion of a measurement.

Following are tables with the error codes and their descriptions.

Parser Errors

Parser errors are normally fatal meaning that the measurement will not take place until the error is corrected.

Error Number	Description
0	No error
1	Illegal clause or field
9	Mode clause or window clause error
10	Mode clause syntax error
11	Mode clause syntax error
12	Trigger clause error
13	trig_x_d (external trigger) error
14	trig_r error
15	Set-up numerical error
16	Accessory type error
17	Accessory index exceeded
20	Mode not allowed
21	Field terminator error in set-up clause
22	Front filter error (not used in 704/714)
23	Rear filter error (not used)

29	Window S (starting wavelength) field error
33	Execute clause field error
37	Window S/E field error
38	Window S field error
39	Window E field error
40	Window E field error
41	Window E field error

Error Number	Description
42	Window clause error
43	Window field error
48	Detector field error
49	Averaging field error
50	Integration field error
51	Not used
56	Command line error
62	Calibration error
63	Mode m_cnt 0
64	Window clause error
65	Trigger clause error

69	Execute clause error
73	Calculation clause error
79	Calculation syntax error
80	Calculation get error
81	Mode illegal
82	Invalid l_d_flag value (calculate clause)
83	Invalid z_flag value (calculate clause)
85	Calculation clause error
91	Calculation field error
98	Measurement mode error

Measurement Errors

Error Number	Description
1001	Data quality unknown.
1002	Overload - Max A/D converter counts at minimum integration time
1003	Low signal.
1004	Operator abort
1005	Hardware malfunction
1006	Program abort
1007	Unknown abort
1008	Measurement time out
1009	Adaptive measurement too long
1010	Too many changes of direction and amplitude during adaptive measurement. Indicates rapidly changing light source.
1020	Light - Dark value too small for calculation

Appendix B
Sample Source Code Listings

Overview

Following are listings of source code found in files on the **\tools\examples** directory, with the extension **.C**. They are written in the "C" programming language and are commented as much as is practical. The user is free to use any or all of the source code(s) in application programs.

Listing 1 - PTRDEM.C

```
/* File - PTRDEM.C  Pointers demonstration.....

This program illustrates a method of running a large
group of measurements and saving them temporarily in
memory using pointers. When the number of
measurements has been performed, the system will then
retrieve each measurement and calculate the
corresponding corrected data set, which is then summed
in a temporary array for averaging. When the
calculations are complete, the resultant value is then
placed in an output file. The purpose of this routine is to
illustrate a method of using pointers.bin, and to illustrate
a method of making multiple measurements.

The user is invited to use fragments of this program when
developing a pointers program. The scheme for getting
the pointers structure loaded and into use, is especially
useful.

    Using Microsoft v5.1  large model.

    cl /AL ptrdem.c

This program uses dos function(48) to establish memory.
It also uses function (36) to automatically locate the ram
disk.  The compiler is Microsoft Vers 5.10

    05-02-90   phm    updated
    01-05-90   phm    updated text.
    09-14-86   phm    Created.
```

```
-------------------------------------------------------------*/
#include   <dos .h>                    /*  standard dos defs */

#include   <conio .h>                  /*  declarations for consol io */

#include  <stdio.h>                    /*  std defs for i/o */

#include   <stdlib.h>                  /*  std library declarations */

#include   <string.h>                  /*  std string library
                                           declarations */

#include "tlsstr4.h"                   /*  toolkit structure template */
   struct  mem
   {
   int num_meas;                       /*  number of measurements */
   int *meas;                          /*  pointer to measurements */
   };

typedef struct mem    mem_t;           /*  This is a scheme for typing
                                           structs */

double sum[MAX_V_PIX];                 /*  space to hold running sum
                                           during avg calc */

int answ = 10;                         /*  answer to query return */
```

```
/* defs file for AT bus parameters*/
#define BRIGHT          0X8
#define BLINK           0X80
#define BLK             0
#define BLU             1
#define GRN             2
#define CYN             3
#define RED             4
#define MAG             5
#define BRN             6
#define WHT             7
#define GRAY            8
#define LBLU            9
#define LGRN            10
#define LCYN            11
#define LRED            12
#define LMAG            13
#define YEL             14
#define BWHT            15
int qclr, nclr;                 /* colors for query routine */
struct pointers sptr, *spptr;
```

```
/* ---------------------------------------------------------
FUNCTION DECLARATIONS */
 void main(void );
void first_meas(void );
void get_pointers(void );
void cls(void );
void atprn(int row,int col,char *str,int clr);
void at(int row,int col);
int atresp(int row,int col,char *str,int siz,int clr,int qclr);
void atint(int row,int col,int *valptr,int radix,int fsiz,int clr,int qclr);
int x_stoi(char *instr,int radix);
/*************************************************
main()
*************************************************/
void main()
{
int    base, segval, i, j, paras;
union REGS inregs, outregs;
mem_t  meas_mem;
unsigned int *dest, *tptr;
float *fptr1, *fptr2, div_val;
double *dptr1;
float  stime, etime, dtime, ttime;          /*floats to hold
start,end,dif, unit test time */
qclr = (BRN < 4) + YEL;
nclr = YEL;                                 /* set up some colors */
cls();                                      /* clear screen */
atprn(3,20," POINTERS DEMONSTRATION ",qclr);
printf("\r\n\n  This routine will make multiple measurements and
store");
```

```
printf("\r\n the results in ram memory, which has been dynamically
allocated.");

printf("\r\n\n You may make any number of measurements up to
500. When");

printf("\r\n the measurement group is complete, the program will
then ");

printf("\r\n calculate corrected data for each measurement, and
calculate");

printf("\r\n an average for each of the 201 virtual data points. This
result");

printf("\r\n is loaded into the corrected structure and can be viewed
by");

printf("\r\n typing correctd.asc.");

printf("\r\n\n\nHit any key to start");

(void)getch();
/*.....
1 -    Get memory allocation. We will require 32 paragraphs per
       measurement, we can allocate all of the memory available if we
       need to. Example 256k dividedby 16 = 16000 paragraphs. which
       divided by 32 = 500 measurements.
.....*/
  outregs.x.cflag = 1;                    /*  funct 48 signals status thru
                                              carry flag */

  while( outregs.x.cflag)

   {
     cls();                               /    *clear screen */

     do
   {

       atprn(5,10," How many measurements ? ",qclr); /* print at row
                                                        & column */

       atint(5,40,&answ,10,4,nclr,qclr); /* get an integer for an
                                            answer */
     }while(answ==0);

     meas_mem.num_meas = answ;  /*  stuff number of meas
                                      here */
     paras = answ * 32;                   /*  and paragraphs here */
```

```
inregs.x.ax = (unsigned)0x4800;          /* Microsofts scheme for int 21
function */

inregs.x.bx = (unsigned)paras;           /* calls. We aredynamically
                                            allocating */

(void)intdos(&inregs,&outregs);          /* memory here, then checking
                                            for ok */

at(7,10);

if(outregs.x.cflag)

{
      if(outregs.x.ax = = 7)

      {

            printf("Memory control block damaged");

           exit(0);

      }

      if(outregs.x.ax = = 8)

      {

      printf("Not enough memory, space available = %d

      measurements",outregs.x.bx / 32);

        }

   }

  }

  base = segval = (int)outregs.x.ax;   /* base of allocated memory */

 /*.....
2 -    Next run the required number of measurements. Since we want to
       work in paragraphs, we will calculate the pointer each time and
       move a rawlight data block to allocated memory. We will keep
       one dark, and then save multiple  rawlights.  We will also print a
       star * for each measurement so you can see progress.
       .....*/

 /* get start time in seconds */

 inregs.x.ax = (unsigned)0x2c00;

 (void)intdos(&inregs,&outregs);         /* call dos 21 funct */
```

```
 stime =(float)( (outregs.h.ch * 3600.0) + (outregs.h.cl * 60.0) +
outregs.h.dh + (outregs.h.dl / 100.0)) ;

 /* open pointers and use the pointers method when making
measurements */

get_pointers();

printf("\r\n*");                          /* print star for first meas */

sptr.pcmd-wn[0].l_d_flag = 0;              /* Set flag to do light
                                           measurements only */

for( i = 0; i  meas_mem.num_meas; i+ +)

{
      (*sptr.emeas)();                     /* make a measurement using
                                           pointers */

      tptr = (unsigned int *) &dest;       /* initializing a pointer (*dest) */

      tptr[0] = 0;                         /* build offset */

      tptr[1] = (unsigned)segval;          /* and segment */

      memcpy(dest,&sptr.pld-data[0],MAX_R_PIX); /* copy result
                                                 to allocated
                                                 mem */

      segval + = 32;                       /* update the paragraph for next
                                           meas */

      printf("*");                         /* and print a star for this
                                           iteration */

  }
  /* get end time in seconds */

  inregs.x.ax = (unsigned)0x2c00;

  (void)intdos(&inregs,&outregs);          /* call dos 21 funct */

etime = (float)((outregs.h.ch * 3600) + (outregs.h.cl * 60) +
outregs.h.dh + (outregs.h.dl / 100.0)) ;

   dtime = etime - stime;                  /* difference in time */

  ttime = dtime / meas_mem.num_meas;

 printf ("\r\n elapsed time = %f avg. test time =  %f",dtime,ttime);
```

```
printf("\r\n Measurements are complete, now calculating corrected
average");

  printf("\r\n This takes a little time so we will print a star ");

  printf("\r\n for each array summed.");

  printf("\r\n\n");

/*.....................................................................
3 -     Now calculate the corrected values for each measurement, and
        sum it in a temporary buffer.  Next we will take the average of
        the sum and load it back to correctd, where it can be read by
        the toolkit after exiting this program.  */
/* get start time in seconds */

inregs.x.ax = (unsigned)0x2c00;

(void)intdos(&inregs,&outregs);              /* call dos 21 funct */

stime =(float)( (outregs.h.ch * 3600) + (outregs.h.cl * 60) +
outregs.h.dh + (outregs.h.dl / 100.0)) ;

  for( i = 0; i  meas_mem.num_meas; i+ +)
  {
   segval - = 32;

   tptr = (unsigned int *) &dest;

   tptr[0] = 0;

   tptr[1] = (unsigned)segval;

   memcpy(&sptr.pld- > data[0],dest,MAX_R_PIX);

   sptr.pmm-corr_flag = FALSE;          /* reset corrected flag */

   (*sptr.xcorrct)();                   /* call corrected */

   dptr1 = &sum[0];

   fptr2 = &sptr.pcd- > data[0];

   for( j = 0; j  MAX_V_PIX; j+ +)

    *dptr1+ +  + = *fptr2+ +;

   printf("*");

  }

   fptr1 = &sptr.pcd- > data[0];
```

```
    dptr1 = &sum[0];
    div_val= (float)meas_mem.num_meas;
    for( i = 0; i  MAX_V_PIX; i++)
    {
     *fptr1++ = (float)(*dptr1++ / div_val);
    }
   /* get end time in seconds */
   inregs.x.ax = (unsigned)0x2c00;
   (void)intdos(&inregs,&outregs);          /* call dos 21 funct */
etime = (float)((outregs.h.ch * 3600) + (outregs.h.cl * 60) +
outregs.h.dh + (outregs.h.dl / 100.0)) ;
   dtime = etime - stime;                   /* difference in time */
   ttime = dtime / meas_mem.num_meas;
printf ("\r\n elapsed time = %f  avg. unit calc time =
%f",dtime,ttime);
   printf("\r\n Averaging done -see corrected.asc for result");
}
/*****************************************************
void first_meas()
*****************************************************/
void first_meas()
{
FILE *fptr;
   if((fptr = fopen("PHOTO","w")) = = NULL)
   {
     perror("can't open PHOTO");
     abort();
   }
   fputs("WAV; S390.0, E730.0,L2;X*",fptr);/* insure all first mea
                                            get  both light & dark */
```

```
    fclose(fptr);
}
/*****************************************************************
void get_pointers()
*****************************************************************/

void get_pointers()
{
FILE *fptr, *in_a;
int numread, dr;
static char fname[20];
  if(( in_a = fopen("TOOLS.CFG","r")) = = NULL)
  {
    printf("\r\nCan't open TOOLS.CFG...aborting");
    exit(1);
  }
  if ((fscanf ( in_a, "%c ",(char *)&dr)) != 1)
  {
    printf("\r\nCan't read TOOLS.CFG...aborting");
    abort();
  }
     fclose(in_a);
  fname[0] = (char)dr;
  strcat ( fname,":pointers.bin");
  if(( fptr = fopen(fname,"rb")) = = NULL)
  {
    printf("Can't open %s",fname);
    abort();
```

```
    }
    spptr = &sptr;
if (( numread = fread((char *) spptr, sizeof(char *),
NUM_PTRS,fptr)) != NUM_PTRS )
    {
     perror("can't read pointers.bin");
     abort();
    }
}
/*******************************************************
 The following support functions were taken from the 'at' series
library and placed here to make the file entirely independent.
*******************************************************/

/*-----------------------------
 cls() clear screen
-----------------------------*/
void cls()
{
union  REGS inregs;
union  REGS outregs;
    inregs.x.ax = (unsigned)0x0600;
    inregs.x.cx = (unsigned)0;
    inregs.h.bh = 7;
    inregs.h.dh = 24;
    inregs.h.dl = 79;
    (void)int86(0x10,&inregs,&outregs);
}
```

```
/*-------------------------------
 atprn()  print string at X Y.
-------------------------------*/
void atprn(row,col,str,clr)
int row, col, clr;
char   *str;
{
union  REGS inregs;
union  REGS outregs;
   inregs.x.ax = (unsigned)0x0200;
   inregs.h.bh = 0;
   inregs.h.dh = (unsigned char)row;
   inregs.h.dl = (unsigned char)col;
   (void)int86(0x10,&inregs,&outregs);
   while(*str)
   {
      inregs.h.ah = 0x09;
      inregs.h.al = (unsigned char)*str++;
      inregs.h.bh = 0;
      inregs.h.bl = (unsigned char)clr;
      inregs.x.cx = (unsigned)1;
      (void)int86(0x10,&inregs,&outregs);
      inregs.x.ax = (unsigned)0x0200;
      inregs.h.bh = 0;
      inregs.h.dh = (unsigned char)row;
      inregs.h.dl = (unsigned char)++col;
      (void)int86(0x10,&inregs,&outregs);
   }
```

```
}

/*-----------------------------
 at() -- goto X Y simple.
-----------------------------*/

void at(row,col)

int row, col;

{

union  REGS inregs;

union  REGS outregs;
        inregs.x.ax = (unsigned)0x0200;

        inregs.h.bh = 0;

        inregs.h.dh = (unsigned char)row;

        inregs.h.dl = (unsigned char)col;

        (void)int86(0x10,&inregs,&outregs);
}
/*-----------------------------

int atresp(row,col,str,siz,clr,qclr) -- at response.
        int row;   row

        int col;   column

        char *str; string pointer

        int siz;   field size

        int clr;   string color;

        int qclr;  query color;
   returns 0 if no response else 1 if any response.

-----------------------------*/

int atresp(row,col,str,siz,clr,qclr)

int row, col, siz, clr, qclr;

char   *str;

{

union  REGS inregs, lcregs;
```

```
union  REGS outregs;
char   *cptr;
char   cstr[80];
  lcregs.x.ax = (unsigned)0x200;
  lcregs.h.bh = 0;
  lcregs.h.dh = (unsigned char)row;
  lcregs.h.dl = (unsigned char)col;
  (void)int86(0x10,&lcregs,&outregs);          /* go to row col */
  inregs.x.ax = (unsigned)0x0920;
  inregs.x.bx = (unsigned)qclr;
  inregs.x.cx = (unsigned)siz;
  (void)int86(0x10,&inregs,&outregs);          /* show field size */
  siz++;                                       /* increment size to
                                                  allow for \0
                                                  termination */
  atprn(row,col,str,qclr); /* output default response */
  (void)int86(0x10,&lcregs,&outregs);          /* go to row col */
  cstr[0] = (char)siz;
  cptr = cgets(cstr);                          /* get new string */
  (void)int86(0x10,&lcregs,&outregs);          /* go to row col */
  inregs.x.ax = (unsigned)0x0920;
  inregs.x.bx = (unsigned)clr;
  inregs.x.cx = (unsigned)siz;
  (void)int86(0x10,&inregs,&outregs);          /* blank default
                                                  response */
  if(cstr[1])
  {
    strcpy(str,cptr);
```

```
        atprn(row,col,str,clr);                    /* put in new
                                                      response */

        return(1);
      }
      else
      {
        atprn(row,col,str,clr);
        return (0);
      }
}
/*-----------------------------------------------------------------
atint(row,col,val,radix,fsiz); get integer input response.
-----------------------------------------------------------------*/
void atint(row,col,valptr,radix,fsiz,clr,qclr)
int row, col, radix,fsiz, clr, qclr;
int *valptr;
{
char   buf[21];
int    val = *valptr;
   (void)itoa(val,buf,radix);
   (void)atresp(row,col,buf,fsiz,clr, qclr);
   *valptr = x_stoi(buf,radix);
}
```

```
/*****************************************************
x_stoi.c More powerful version of atoi pass radix on call for use with
at functions
*****************************************************/

int x_stoi(instr, radix)

char  *instr;

int radix;

{
      /*Convert string to integer. If string starts with 0x it it  interpreted as a
      hex number, else if it starts with a O it is
      octal, else it is decimal. Conversion stops on encountering
      the first character which is not a digit in the indicated radix.
      *instr is updated to point past the end of the number  */

  register int   num = 0;

  char  *str;

  register int   sign = 1;

  str = instr;

while(*str = = ' ' || *str = = '\t' || *str = = '\n' ) /* skip white
space */

    str + + ;

  if( *str = = '-' )

  {

    sign = -1 ;

    str + + ;

  }

  if(radix  = = 16 )

  {

      while( ('0'  < = *str && *str < = '9') ||

             ('a' < = *str && *str < = 'f') ||

             ('A' < = *str && *str < = 'F') )

      {
```

```
            num *= 16;
            num += ('0' <= *str && *str <= '9') ?
              *str - '0'          :
              toupper(*str) - 'A' + 10;
            str++;
        }
    }
    else if (radix == 8)
    {
      while( '0' <= *str && *str <= '7' )
      {
        num *= 8;
        num += *str++ - '0';
      }
    }
    else
    {
      while( '0' <= *str && *str <= '9' )
      {
          num *= 10;
          num += *str++ - '0';
      }
    }
    instr = str;
    return( num * sign );
}
```

Listing 2 - SPEED.C

```
/*-----------------------------------------------------------------

File - SPEED.C  Pointers demonstration.....

This program illustrates a method of running a large
group of measurements for the purpose of getting
relative speed comparisons.

The compiler is Microsoft Vers 5.10

  05-02-90   phm   update.

  10-02-86   phm   Created.

-----------------------------------------------------------------*/
#include  <dos .h>              '/* standard dos defs */
#include  <conio .h>            /* declarations for consol io */
#include  <stdio.h>             /* std defs for i/o */
#include  <stdlib.h>            /* std library */
#include  <string.h>            /* std string library */
#include  "tlsstr4.h"           /* toolkit structure template */
int answ;                       /* answer to query return */
/* defs file for AT bus
*/
#define BRIGHT                  0X8
#define BLINK                   0X80
#define BLK                     0
#define BLU                     1
#define GRN                     2
```

```
#define CYN             3
#define RED             4
#define MAG             5
#define BRN             6
#define WHT             7
#define GRAY            8
#define LBLU            9
#define LGRN            10
#define LCYN            11
#define LRED            12
#define LMAG            13
#define YEL             14
#define BWHT            15
int qclr, nclr;         /* colors for query routine */
struct pointers sptr, *spptr;
```

```
/* ------------------------------------------------------------
FUNCTION DECLARATIONS
------------------------------------------------------------ */
void initiate(), first_meas(), get_pointers();
void atint(
        int row,                    /* row location */
        int col,                    /* column location */
        int *valptr,                /* value string pointer */
        int radix,                  /* conversion radix */
        int  fsiz,                  /* field size */
        int  clr,                   /* color */
        int  qclr                   /* query color */
        );
int atresp( int  row,               /* row location     */
        int  col,                   /* column location  */
        char *str,                  /* string pointer   */
        int  siz,                   /* field size       */
        int  clr,                   /* color            */
        int  qclr                   /* query color      */
      );
int x_stoi(char *instr,             /* in string */
         int  radix                 /* conversion radix */
        );
void atprn(int row,                 /* row location */
        int col,                    /* column location */
        char *str,                  /* string pointer */
        int clr                     /* color */
        );
```

```
void at(int row,                                    /* row location */
int col                                             /* column location */
 );
void cls(void);
void get_pointers(void);
void first_meas(void);
void main(void);
/*************************************************************
main()
*************************************************************/

void main(void)
{
int    i;
union REGS inregs, outregs;
float  stime, etime, dtime, ttime;  /* floats to hold start,end,dif, unit
test time */
```

```
/*.....
1. Get number of measurements.
.....*/

   qclr = (BRN < < 4) + YEL; nclr = YEL;   /* set up some colors
                                              */

   cls();                                  /* clear screen */

   atprn(3,20," POINTERS DEMONSTRATION ",qclr);

   printf("\r\n\n  This routine will make multiple measurements");

   printf("\r\n and calculate corrected + cie each time.");

   printf("\r\n\n\nHit any key to start");

   (void)getch();

   cls();                                  /* clear screen */

   atprn(5,10," How many measurements ? ",qclr); /* print at row &
                                                    column */

   atint(5,40,&answ,10,4,nclr,qclr);       /* get an integer for an
                                              answer */

/*.....
 2. Next run the required number of measurements.
.....*/

/*  make first measurement thru PHOTO  to establish struct
command  */

   first_meas();

   /* open pointers and use the pointers method when making
measurements */

   get_pointers();

   /* get start time in seconds */

   inregs.x.ax = (unsigned) 0x2c00;

   (void)intdos(&inregs,&outregs);         /* call dos 21 funct */

   stime = (outregs.h.ch * 3600) + (outregs.h.cl * 60) + outregs.h.dh
 + (outregs.h.dl / 100.0) ;

   for( i = 0; i < answ; i + + )

   {
```

```
    (*sptr.emeas)();                    /* make a measurement
                                           using pointers */

    (*sptr.xcorrct)();

    (*sptr.xciecalc)();

    printf("*");

   }

   /* get end time in seconds */

   inregs.x.ax = (unsigned) 0x2c00;

   (void)intdos(&inregs,&outregs);      /* call dos 21 funct */

etime = (outregs.h.ch * 3600) + (outregs.h.cl * 60) + outregs.h.dh
+ (outregs.h.dl / 100.0) ;

   dtime = etime - stime;               /* difference in time */

   ttime = dtime / answ;

printf ("\r\n elapsed time = %f avg. test time = %f",dtime,ttime);

printf("\r\n Measurements are complete, now calculating corrected
average");

}
```

```
/*************************************************************

void first_meas()

*************************************************************/
void first_meas(void)

{

FILE *fptr;

  if((fptr = fopen("PHOTO","w")) = = NULL)

  {

    perror("can't open PHOTO");

    abort();

  }

  fputs("WAV; S390.0, E730.0,L2;X*",fptr); /* insure all first meas
get both light & dark */

  fclose(fptr);

}

/*************************************************************

void get_pointers()

*************************************************************/
void get_pointers(void)

{

FILE *fptr;

union REGS inregs,outregs;

int i, numread, dr;

static char fname[20];

#define MAXDRV 10

/*  the following loop determines which drive is the ram disk
automatically. We are looking for a disk which uses 1 sector per
allocation unit, and ECh (236d) allocation units total.

 */

   for ( i = 3; i < MAXDRV ; i+ + )
```

```
    {
      inregs.x.ax = (unsigned) 0x3600;    /* function 36 get disk free
space*/
      inregs.h.dl = (unsigned char)i;
      (void)intdos(&inregs, &outregs );
      if ( outregs.x.ax != 1 )
        continue;
      if ( outregs.x.dx == 0x16a )
        break;
    }
   if ( i == MAXDRV )
   {
     printf("couldn't find virtual disk ...exiting");
     exit(2);
   }
   dr = 0x40 + i;
   printf("\r\n virtual drive = %c\r\n",dr);

   fname[0] = (char)dr;
   strcat ( fname,":pointers.bin");
   if(( fptr = fopen(fname,"rb")) == NULL)
   {
    perror("can't open pointers.bin");
    abort();
   }
   spptr = &sptr;
 if (( numread = fread((char *) spptr, sizeof(char *),
 NUM_PTRS,fptr)) != NUM_PTRS )
    {
```

```
    perror("can't read pointers.bin");
    abort();
   }
}
/*****************************************************************

 The following support functions were taken from the 'at' series
library and placed here to make the file entirely independent.
*****************************************************************/

/*-----------------------------
 cls() clear screen
-----------------------------*/
void cls(void)
{
union  REGS inregs;
union  REGS outregs;
   inregs.x.ax = (unsigned) 0x0600;
   inregs.x.cx = (unsigned) 0;
   inregs.h.bh = 7;
   inregs.h.dh = 24;
   inregs.h.dl = (unsigned char)79;
   (void)int86(0x10,&inregs,&outregs);
}
/*-----------------------------
 atprn()  print string at X Y.
-----------------------------*/
void atprn(int row,                    /* row location */
      int col,                         /* column location */
      char *str,                       /* string pointer */
      int clr                          /* color */
```

```
    )
{
union  REGS inregs;
union  REGS outregs;
  inregs.x.ax = (unsigned) 0x0200;
  inregs.h.bh = 0;
  inregs.h.dh = (unsigned char)row;
  inregs.h.dl = (unsigned char)col;
  (void)int86(0x10,&inregs,&outregs);
  while(*str)
  {
    inregs.h.ah = 0x09;
    inregs.h.al = (unsigned char)*str++;
    inregs.h.bh = 0;
    inregs.h.bl = (unsigned char)clr;
    inregs.x.cx = (unsigned) 1;
    (void)int86(0x10,&inregs,&outregs);
    inregs.x.ax = (unsigned) 0x0200;
    inregs.h.bh = 0;
    inregs.h.dh = (unsigned char)row;
    inregs.h.dl = (unsigned char)++col;
    (void)int86(0x10,&inregs,&outregs);
  }
}
```

```
/*-----------------------------
 at() -- goto X Y simple.
-----------------------------*/
void at( int row,                                   /* row location */
      int col                                       /* column location */
    )
{
union  REGS inregs;
union  REGS outregs;
     inregs.x.ax = (unsigned) 0x0200;
     inregs.h.bh = 0;
     inregs.h.dh = (unsigned char)row;
     inregs.h.dl = (unsigned char)col;
     (void)int86(0x10,&inregs,&outregs);
}
/*-----------------------------------------------------------------
int  atresp(row,col,str,siz,clr,qclr) -- at response.
  int row;   row
  int col;   column
  char *str;  string pointer
  int siz;   field size
  int clr;   string color;
  int qclr;  query color;
  returns 0 if no response else 1 if any response.
-----------------------------------------------------------------*/
atresp( int  row,                                   /* row location    */
      int  col,                                     /* column location  */
      char  *str,                                   /* string pointer   */
```

```
        int  siz,                                   /* field size     */
        int  clr,                                   /* color         */
        int  qclr                                   /* query color   */
      )
{
union  REGS inregs, lcregs;
union  REGS outregs;
char   *cptr;
char   cstr[80];
  lcregs.x.ax = (unsigned) 0x200;
  lcregs.h.bh = 0;
  lcregs.h.dh = (unsigned char)row;
  lcregs.h.dl = (unsigned char)col;
  (void)int86(0x10,&lcregs,&outregs);          /* go to row col */
  inregs.x.ax = (unsigned) 0x0920;
  inregs.x.bx = (unsigned) qclr;
  inregs.x.cx = (unsigned) siz;
  (void)int86(0x10,&inregs,&outregs);          /* show field size */
  siz++;       /* increment size to allow for \0 termination */
  atprn(row,col,str,qclr);                  /* output default response */
  (void)int86(0x10,&lcregs,&outregs);          /* go to row col */
  cstr[0] = (char)siz;
  cptr = cgets(cstr);                          /* get new string */
  (void)int86(0x10,&lcregs,&outregs);          /* go to row col */
  inregs.x.ax = (unsigned) 0x0920;
  inregs.x.bx = (unsigned) clr;
  inregs.x.cx = (unsigned) siz;
```

```
    (void)int86(0x10,&inregs,&outregs);    /* blank default response */
    if(cstr[1])
    {
      strcpy(str,cptr);
      atprn(row,col,str,clr);              /* put in new response */
      return(1);
    }
    else
    {
       atprn(row,col,str,clr);
       return (0);
    }
}
/*---------------------------------------
 atint(row,col,val,radix,fsiz);
 get integer input response.
---------------------------------------*/
void atint(
       int row,                          /* row location */
       int col,                          /* column location */
       int *valptr,                      /* value string pointer */
       int radix,                        /* conversion radix */
       int  fsiz,                        /* field size */
       int   clr,                        /* color */
       int  qclr                         /* query color */
            )
       {
       char   buf[21];
int    val = *valptr;
  (void)itoa(val,buf,radix);
```

```
    (void)atresp(row,col,buf,fsiz,clr, qclr);
    *valptr = x_stoi(buf,radix);
}
/*****************************************************************

 x_stoi.c More powerful version of atoi pass radix on call for use with
at functions

*****************************************************************/
int x_stoi(char *instr,                    /* in string */
       int  radix                          /* conversion radix */
      )
{
/*****************************************************************

    Convert string to integer. If string starts with 0x it is     interpreted
as a hex number, else if it starts with a O it is octal, else it is decimal.
Conversion stops on encountering the first character which is not a
digit in the indicated radix. *instr is updated to point past the end of
the number.

*****************************************************************/
  register int   num = 0;
  char  *str;
  register int   sign = 1;
  str = instr;
  while(*str = = ' ' || *str = = '\t' || *str = = '\n' )
 /* skip white space */
   str + + ;
   if( *str = = '-' )
   {
     sign = -1 ;
     str + +;
   }
```

```
if(radix == 16)
{

    while(          ('0' <= *str && *str <= '9') ||
                    ('a' <= *str && *str <= 'f') ||
                    ('A' <= *str && *str <= 'F'))
        {
                    num *= 16;
                    num += ('0' <= *str && *str <= '9') ?
                        *str - '0'          :
                        toupper(*str) - 'A' + 10;
                    str++;
        }
}
else if (radix == 8)
{
        while( '0' <= *str && *str <= '7' )
        {
          num *= 8;
          num += *str++ - '0';
        }
}
else
{
        while( '0' <= *str && *str <= '9' )
        {
            num *= 10;
            num += *str++ - '0';
    }
}
        instr = str;
        return( num * sign );
}
```

Listing 3 - TOOLTST1.C

```
/*****************************************************

Tooltst1.c

This file will test tools by opening POINTERS.BIN
intalling a loaded struct pointers, and then making calls
directly via pointers to various operations.

Note that to set up the system, the first call must be made
through PHOTO.

Also.. the drive letter is hard coded in the fopen calls, to
keep things simple.

  01/05/90 phm   update text.

  06/30/89 phm   update to load via tools.cfg

  05/22/86 phm   created
```

```
*****************************************************/
#include                    <stdio.h>
#include                    <stdlib.h>
#include                    <string.h>
#include   "tlsstr4.h"
struct pointers   sptr, *spptr;
char   resp[10];
/*****************prototype declarations ********************/
void main(void );
void get_pointers(void );
/*************************************************/
void main()
{
      int i;
  /*---------------------------------------
  Open pointers, and fill the structure
  ---------------------------------------*/
  get_pointers();
  if(sptr.pmm->s_ver != STRU_VERS)
  {
        printf ( "\r\n ERROR... Toolstru mismatch ...aborting");
         exit(1);
  }
  /* ---------------------------------
    set aperture control code to  0 (leaves aperture in manually
    set position

    possible values              1 = aperture a.
                                 2 = aperture b.
   ---------------------------------*/
  sptr.pcmd-aper_code = 0;
```

```
    /* ------------------------------------
     make a measurement
    ------------------------------------*/
       (*sptr.emeas)();                    /* call measurement */
    /* ------------------------------------
     call the calculation routine
    ------------------------------------*/

    (*sptr.xcorrct)();                     /* call corrected calc */

    /* ------------------------------------
     print results of calculation
    ------------------------------------*/

    for ( i = 0; i  MAX_V_PIX; i+ + )

    {
            printf("\r\n%13.6E",sptr.pmm- > md.cd.data[i]);
    }

}
/*****************************************************************
void get_pointers()
*****************************************************************/
void get_pointers()

{

FILE *fptr, *in_a;

int numread, dr;

static char fname[20];

   if(( in_a = fopen("TOOLS.CFG","r")) = = NULL)

   {
      printf("\r\nCan't open TOOLS.CFG...aborting");
      exit(1);
   }

   if ((fscanf ( in_a, "%c ",(char *)&dr)) ! = 1)

   {
      printf("\r\nCan't read TOOLS.CFG...aborting");
      exit(1);
   }
         fclose(in_a);
```

```
    fname[0] = (char)dr;
    strcat ( fname,":pointers.bin");
    if(( fptr = fopen(fname,"rb")) = = NULL)
    {
        perror("can't open pointers.bin");
        abort();
    }
    spptr = &sptr;
        if (( numread = fread((char *) spptr, sizeof(char *),
        NUM_PTRS, fptr)) != NUM_PTRS )
        {
        perror("can't read pointers.bin");
        abort();
        }
}
```

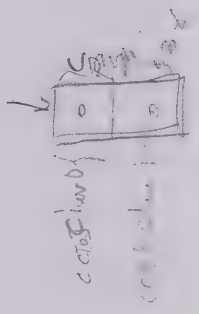
Listing 4 - TOOLTST2.C

```
/*******************************************************
Tooltst2.c --

This file will test tools by providing a command line and
then checking the returned errorcode through the
character device ("PHOTO"). It will then open
CORRECTD.BIN, get the values and print them out.

  1-05-90 phm update text.

  5/22/86 phm created

*******************************************************/
#include                    <stdio.h>
#include                    <stdlib.h>
#include                    <string.h>
#include   "tlsstr4.h"
struct meas mmeas, *mm;
FILE  *fptr, *in_a;
FILE  *file_ptr;
char  fname[20];
char  resp[10];
char  dr;
/*****************function prototypes ************/
void main(void );
int get_bin_fil(char *name,char *buff);
/*************************************************/
```

```
void main()
{
int i;
  /* ------------------------------------
    Open PHOTO and make a measurement
   ------------------------------------*/
  if((fptr = fopen("PHOTO","w")) = = NULL)
  {
       perror("can't open PHOTO");
       abort();
  }
  fputs("X*",fptr);
  fclose(fptr);
  if((fptr = fopen("PHOTO","r")) = = NULL)
  {
      perror("can't open PHOTO");
      abort();
  }
  printf("\nErrorcode = %s",(fgets(resp,10,fptr)));
  fclose(fptr);
  /* ------------------------------------
    Open tools.cfg and get drive data
   ------------------------------------*/
  if(( in_a = fopen("TOOLS.CFG","r")) = = NULL)
  {
      printf("\r\nCan't open TOOLS.CFG...aborting");
      abort();
  }
  if ((fscanf ( in_a, "%c ",&dr)) ! = 1)
  {
      printf("\r\nCan't read TOOLS.CFG...aborting");
      abort();
  }
```

```
        fclose(in_a);
    /* ------------------------------------
      Open CORRCTD.BIN and read in data
     ------------------------------------*/

    mm = &mmeas;

    fname[0] = dr;

    strcat ( fname,":correctd.bin");
        if((get_bin_fil( fname,(char *)&mm->md.cd)) == NULL)

        {

        perror("error correctd.bin");

        abort();

        }
    /* ------------------------------------
      Print out the data.
     ------------------------------------*/
         for( i=0; i MAX_V_PIX; i++)

           printf("\r\n%13.6E",mm->md.cd.data[i]);
}

/*************************************************
Sub-routine to read a binary file.

int get_bin_fil ( name, buff )
*************************************************/

int get_bin_fil ( name, buff )

/*----------------------------------------------------------------
get a binary file that is an image of a structure function.  1st 2
bytes of file must be binary length of file in bytes.  Returns pointer to
buffer. Automatically allocates the size.
----------------------------------------------------------------*/

char *name;                    /* name of file to get */

char *buff;                    /* target buffer */

{

 union
        {

        unsigned length;

        char cc[2];
```

```
        } uu;

if (( file_ptr = fopen ( name, "rb" )) == NULL )
        {
        printf("trouble opening");
        return NULL;
        }
if ( fread ( uu.cc, 2, 1, file_ptr ) != 1 )
        {
        fclose ( file_ptr );
        return(NULL);
        }
        buff[0] = uu.cc[0];
        buff[1] = uu.cc[1];
if ( fread ( buff + 2, 1, uu.length-2, file_ptr ) != uu.length-2)
        {
        fclose ( file_ptr );
        return NULL;
        }
fclose ( file_ptr );
return(1);
}
```



Appendix C
TOOLSTRU

Overview

TOOLSTRU (found in file **TLSSTR4.H**) is the master template of all structures and functions used by Toolkit except **struct hd_stat** (found in file **PR704STR.H**) which defines **STATUS.ASC** and **STATUS.BIN.**

All references to SPATIAL and or POS are not used by this version of Toolkit.

A listing of **PR704STR.H** follows the **TOOLSTRU** listing.

This listing has been slightly modified to aid in readability. No definitions of pointers or functions have been changed.

Listing 1 - TOOLSTRU

```
/*

   File:  tlsstr4.h    (previously named toolstru.h)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  Copyright (C) 1988 Photo Research Division of
Kollmorgen Corp.

  All rights reserved.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

This file serves as the basic template for a number of
different programs
```

NOTE:
It is mandatory that all programs track to the same version of TOOLSTRU. If any changes are made to toolstru which could cause a mismatch problem, then 'STRU_VERS' must be changed.

This file is based on file ucistru.h

This file contains the following structures

cmd_wndw

result_wndw

calc_w

pks_vls

calc_s

meas_data

meas

command

accessory

code_indx_tbl

limits

status

adapt

calib

spectral

spare

pointers

08-10-90 r.k. new STRU_VERS

08-01-90 r.k. changed meaning of struct acc.cal_flags[]

07-16-90 r.k. revised struct calib to have rel calib coe separate from v_pix mapping

05-14-90 phm revised pointers.

05-01-90 r.k. MAX_ACC was 6 now 12 added adapt time lim for "low light"

02-16-90 TLC Added QUAL_WK for SPR 70

01-30-90 r.k. new version also for 380..780 nm

11-30-88 r.k. renamed tlsstr4.h

```
  09-13-88  phm   pulled spatial from pointer list..

  04-10-88  r.k.  NEW VERSION OF THESE
STRUCTS!!!!!!!!!!!!!!!!!

  03-31-88  r.k.  added TRUE,FALSE,PR704

  03-09-87  phm   added adp time error. TIME_ABORT

  01-24-87  phm   Vers 01870 added cmd-meas_mode for
pulse capability. This forces vers 4.0 change list of changes prior
to 01-01-87 were deleted for brevity (r.k.)

*/

/*-----------------------------------------------------------------*/

#define TOOLSTRU 0

#define STRU_VERS  8900     /* VERS mo/year incr number

                               Number not to exceed 32767 */

/*.................................................*/

#define TRUE 1

#define FALSE 0

#define IOBASE 0x260          /*  default i/o base address of
                                  head */

#define MAX_R_PIX 256         /*  max number of raw pixels */

#define MAX_V_PIX 220         /*  max num of virtual pixels  */

#define MAX_DIM 4             /*  max number of dimensions of
                                  main data array  */

#define M_NAME_SZ  32         /*  max size in bytes of meas
                                  description */

#define M_EXPN_SZ  64         /*  num bytes reserved for
                                  expansion of struct */

#define M_NUM_WNDW 1          /*  max number of measurement
                                  windows */

#define S_NAME_SZ 32          /*   max number of bytes (incl
                                   terminator) of spectral data
                                   descriptor */
```

```
#define S_EXPN_SZ 32          /*  number of bytes in spectral
                                  data structure reserved for
                                  expansion */

#define N_AUX_DATA 16         /*  number of auxiliary floating
                                  point data values associated
                                  this structure  */

#define MAX_NUM_ACCSRY 12 /* number of accessories
                             of all types front & rear
                             filters + accessories */

#define MAX_NUM_LENS  10      /* number of accessory
                                 lenses */

#define ACC_NAME_SZ 14    /*  max size in bytes of
                              accessory description*/

#define ACC_FILE_NM_SZ 14 /*  max size in bytes of
                              accessory spectral data file  */

#define MAX_APER 2            /* maximum number of apertures
                                 */

#define APER1 1

#define APER2 2

#define NUM_T_FACT 14     /* number of integration time
                             correction factors */

#define NUM_MODES 2       /* number of modes of
                             operation */

#define ALLOWD 0

#define NOT_ALWD 1

#define CALIBRATED 2

#define UNCALIB 3

#define WAV_INDX 0

#define POS_INDX 1

                          /* return codes for meas_func ()
                             */

#define QUAL_OK 0   /* okay */

#define QUAL_QQ 1             /* questionable quality */
```

```
#define QUAL_OV 2          /*  A to D overflow */
#define QUAL_UN 3          /*  signal too small */
#define OPER_ABORT 4       /*  meas aborted by operator */
#define MALFNCT 5          /*  hardware malfunction */
#define PROG_ABORT 6       /*  meas aborted by meas_trap()
                               */
#define UNKN_ABORT 7       /*  something is wrong but I can't
                               put my  finger on it */
#define TIME_ABORT 8       /*  meas operation timed out */
#define TOO_LONG 9         /*  total time in meas adapt too
                               long */
#define FLIPS 10           /*  too many changes of direction
                               of integr time in adapt meas.
                               probably light source is
                               varying */
#define QUAL_WK 20         /*  Light minus dark to small for -
                               calculations  SPR 70 TLC */

#define LOWPRES 1
#define LIM_OK 1
#define LIM_NRW 0
#define NOT_CALC 0
#define C_683 1
#define C_100R 2
#define SPECT 0
#define SPATL 1
#define SPATL2D 11
#define NORML 0
#define RAD 0
#define IRRAD 1
#define RADINT 2
#define NA 9
```

```
#define LUMC 0
#define ILLUM 1
#define LUMINT 2
#define UNCAL 3
#define RAW  4
#define WAV 7
#define FACTRY 0
#define CUST 1
#define ADAPT 9999
#define RET128 3
#define RET256 4
#define RET256X256 5
#define DC 0
#define POS 0
#define NEG 1
#define POS_NEG 2
#define NANO 0
#define MICRO 1
#define MILS 2
#define MM  3
#define DEGR 0
#define MILL_RAD 1
#define D_ONLY 4
#define RDY 0                    /* trigger options */
#define MSG 1
#define EXT 2
#define DELAY 3
#define TOD 4
```

```
#define OPEN 1
#define CLOSED 0
#define FOPEN 5
#define ROPEN 0
#ifdef tlctlc900503
#define MANUAL 2
#define UTIL 0
#define AUTO 1
#define CPU 1
#endif
#define MEAS 1
#define ZERO 0
#define VIEW 1
#define FAST 1
#define NOTFAST 0
#define LOW 1
#define HIGH 0
#define ENGLISH 0
#define METRIC 1
#define PR701 0
#define PR703 1
#define PR711 2
#define PR713 3
#define PR719 4
#define PR704 5
#define PR714 6
#define PULSE 0
#define MOTOR 1
```

```
#define POWER 0
#define ENERGY 1
#define PAUSE 2
#define RAPID 3
#define MOVING_SHTR 2  /* slave immediate status mask bits */
#define MEASURING 4
#define PROC_CMD 8
#define FAULT 0x10
#define SYNC_OUT 0x80
/*-----------------------------------------------------------------*/
struct cmd_wndw
  {
                                    /*  Note - Window [0] refers to
                                        the overall limits of the
                                        measurement */
float start_w_p, end_w_p, w_p_incr; /* spectral or spatial
                                        boundaries and interval for
                                        this window */
 float bandwidth;                   /*  bandwidth for this window  */
 float integr_speed;                /*  integration speed indicator
                                        possible values: .026...30
                                        integration time in seconds
                                        ADAPT adaptive integ time
                                        selection */

    int detector;                   /* detector type
                                    possible values
                                RET128   128 element Reticon
                                RET256   256 element Reticon
                                RET256X256  256 * 256 element Reticon  */
```

```
    int avg_method;          /*  averaging method possible
                                 values ADAPT processor
                                 makes averaging decisions
                                 based on light level and
                                 signal noise 1...100 average
                                 1 to 100 readings */

    int meas_mode;           /*  flag to indicate PR719 meas
                                 mode

                    possible values:
                      POWER
                      ENERGY
                      PAUSE
                      RAPID     */

    int l_d_flag;                /* meas light, dark or both flag
                    0 = light only
                    1 = dark only
                    2 = both    */

    int z_flag;    /* close zero shutter during dark meas flag
                    0 = close zero shutter during dark meas
                    1 = leave z shtr open during dark meas*/

  };
/*-------------------------------------------------------------*/

struct result_wndw
{
   /* Note - Window [0] refers to the overall limits of the
measurement */
    int d_qual;    /* data quality indicator possible values:
                      QUAL_OK   quality okay
                      QUAL_QQ   questionable quality
                      QUAL_OV   overflow
                      QUAL_UN   underflow
```

```
                OPER_ABORT   meas aborted by operator
                MALFNCT      meas aborted by head malfunction
                PROG_ABORT   meas aborted by meas_trap()
                             return in adaptive meas
                UNKN_ABORT   meas aborted for unknown reasons */
        int low_range_flag;          /*  low rang flag possible values:
                                         n.a. in PR700 series */
        int polarity_flag;           /*  data range flag possible
                                         values:
                                         n.a. in PR700 series */

    };
/*-------------------------------------------------------------*/
struct calc_w
    {
        int size;        /* size in bytes of this structure  */
        int d_lim;       /* wavelength limits indicator possible values:
                            LIM_OK  limits okay for photometric calc's
                            LIM_NRW limits too narrow for photometry */
        int lum_eff_flag;            /*  luminous efficacy used in CIE
                                         calculations
                                     possible values:
                                    C_683  = 683 Lm/Watt
                                    C_100R = 100 relative */
        float intg_radnc;            /*  integrated spectral rad/irrad/
                                         etc. the meaning of this field
                                         is defined by radnc_flag
                                         see radnc_flag  */
        float photn_radnc;           /*  photon rad/irrad/etc. The
                                         meaning of this field is
                                         defined by radnc_flag */
```

```
    int observer;    /* field of view of standard observer for
                        CIE tristimulus tables
                     possible values:

                      2  2 degree

                     10  10 degree  */
    float cie_x;     /* CIE X  (1931)  */
    float cie_y;     /* CIE Y also Cd/m2 or Lux (1931)  */
    float cie_z;     /* CIE Z (1931)  */
    float cie_v;     /* scotopic Cd/m2 or Lux  */
    float cie_ye;    /* same as cie_y but in English units of
                        foot-lamberts or foot-candles */
    float cie_ve;    /* same as cie_v but in English units  */
    float cie_xx;    /* CIE x  (1931)  */
    float cie_yy;    /* CIE y  (1931)  */
    float cie_uu;    /* CIE u  (1960)  */
    float cie_vv;    /* CIE v  (1960)  */
    float cie_vp;    /* CIE v' (1976)  */
  };
/*-----------------------------------------------------------------*/
struct cctemp    /* this structure is the template for the
toolbox binary file format */
 {
    int size;
    float k_color_temp;          /*  correlated color temp in
                                     degrees K */
    float mk_color_temp;         /*  correlated color temp in
                                     MK-1 units */
    float dev_color_temp;        /*  deviation of color temp off
                                     Plankian locus in CIE
                                     u-v units  */
 };
```

```
/*-----------------------------------------------------------------*/
struct int_data
 {
   int size;                    /*  size of this struct */
   long itime;                  /*  integration time in millisec
                                    of this data */
   int num_dim;                 /*  holds number of dimensions
                                    of data below
                  possible values: 1...MAX_DIM    */
   int dim_sz[MAX_DIM];         /*  array holds dimension sizes
                                    of data below */
   int data[MAX_R_PIX];         /*  holds raw data from test
                                    head */
 };
/*-----------------------------------------------------------------*/
struct flt_data
 {
   int size;                    /*  size of this struct */
   long itime;                  /*  integration time in millisec
                                    of this data */
   int num_dim;                 /*  holds number of dimensions
                                    of data below
                  possible values: 1...MAX_DIM  */
   int dim_sz[MAX_DIM];         /*  array holds dimension sizes
                                    of data below */
   float data[MAX_V_PIX];       /*  holds corrected data from
                                    test head */
 };
/*-----------------------------------------------------------------*/
```

```
struct meas_data
 {
  int size;                    /*  size of this structure */
  int pho_units;               /*  photometric units indicator
                                   possible values:
                       ENGLISH     footLamberts, footcandles
                       METRIC      Cd/m2, Lux     */
  int spe_units;               /*  spectral units indicator
                                   possible values:
                       NANO        nanometers
                       MICRO       micrometers    */
  int radnc_flag;              /*  radiance mode flag
                                   possible values:
                       RAD         radiance
                       IRRAD       irradiance
                       RADINT      radiant intensity
                       UNCAL       uncalibrated for accessories but
                                   corrected for bandwidth
                       RAW    raw data  (sensor output) */
  int lumnc_flag;              /* luminance mode flag
                                   possible values:
                       LUMC    luminance
                       ILLUM   illuminance
                       LUMINT  luminance intensity
                       UNCAL   uncalibrated    */
```

```
    int cal_flag;                /* calibrated results flag
                                    possible values:
                                    CALIBRATED   calibrated results
                                    UNCALIB      uncalibrated results through
                                                 use of an accessory not
                                                 calibrated for specified
                                                 mode of operation  */

    int meas_mode;               /*  flag to indicate PR719 meas
                                     mode
                                    possible values:
                                    POWER
                                    ENERGY
                                    PAUSE
                                    RAPID      */
    int  cycles;
    long time;                   /*  integration time of meas in
                                     milliseconds */
    struct int_data ld;          /*  holds raw light data and
                                     params */

    struct int_data dd;          /*  holds raw dark data and
                                     params */

    struct int_data sd;          /*  holds raw light minus dark
                                     data and  params */

    struct flt_data cd;          /*  holds corrected data and
                                     params */

  };
/*-------------------------------------------------------------*/
```

```
struct adapt                    /*  data to support adaptive
                                    sensitivity measuring */
{
int first_raw_pix[MAX_APER];  /* index of 1st raw pixel to
                                    use to find  peak value of a
                                    measurement */

int last_raw_pix[MAX_APER]; /* index of last raw pixel to
                                    use to find peak */

int low_adp_range;              /*  raw data value of low end of
                                    range for peak value */

int high_adp_range;             /*  raw data value of high end of
                                    range for peak value */

int ideal_adp_value;            /*  target raw data value for
                                    peak value */

long low_light_time;            /*  intgration time in ms above
                                    which adapt sens function
                                    returns a "low light" indication.
                                    (note this may be less than
                                    max_i_time) */

long last_i_time;               /*  last integration time in
                                    milliseconds */

int first_i_time_flag;          /*  controls the first integr time
                                    to use in an adaptive measure

                    0 = minimum integration time

                    1 = min time then last integr time if
                      confident no saturation will occur

                    2 = last integration time   */

long total_adp_time;            /*  total adapt measuring time
                                     when exceeded causes an
                                    abort */

};
/*-------------------------------------------------------------*/

struct meas
```

```
{
  int size;                    /*  number of bytes in this
                                   structure */
  int s_ver;                   /*  version of this structure */
  char idate[22];              /*  comp str for idate.exe */
  char descr[M_NAME_SZ]; /*    description of measurement -
                                   ASCII text */
  char date[10];               /*  date of meas - ASCII text in
                                   format yy/mm/dd */
  char time[10];               /*  time of meas - ASCII text in
                                   format hh:mm:ss if hh  23
                                   time is invalid   */
  long b_time;                 /*  time of measurement
                                   completion inmilliseconds
                                   since GMT 00:00:00
                                   Jan 1, 1970 */
  char ser_num[10];            /*  ASCII text serial number of
                                   instrument that made
                                   measurement  */
  long int idnum;              /*   identification number */
  int num_wndws;               /* number of meas windows
                  possible values:
                       0...M_NUM_WNDW       */
  struct cmd_wndw wn[M_NUM_WNDW];
                  /* measurement windows from command */

  struct result_wndw r_wn[M_NUM_WNDW];
                  /* results  indicators per meas window   */
  int aper_code;               /* aperture actually used for
                                  this measurement
                                  APER1..APERMAX_APER  */
```

```
int run_flag;                   /* type of meas run flag
                                   possible values:
                                SPECT spectral run
                                SPATL spatial run  */
int calc_type;                  /* calculation type indicator
                                   possible values:
                                NORML       normal data  */
int meas_flag;                  /*  measurement made and raw
                                    data available flag
                                   possible values:
                                TRUE
                                FALSE  */
int corr_flag;                  /* corrected data calculated flag
                                   possible values:
                                TRUE
                                FALSE  */
int cie_flag;                   /* cie data calculated flag
                                   possible values:
                                TRUE
                                FALSE  */
int cct_flag;                   /* color temperature calculated flag
                                   possible values:
                                TRUE
                                FALSE  */
int fht_flag;                   /* flying head height calculated flag
                                   possible values:
                                TRUE
                                FALSE  */
```

```
int spatl_flag;                 /*  spatial calculations performed
                                    yet
                    possible values:
                TRUE
                FALSE  */
int we_them;                    /* customer or factory flag
                    possible values:
                FACTRY          meas done in factory
                CUST            meas done by customer    */
char aper_desc[14];             /* aperture description */
int num_acc;                    /*  number of external accessories
                                    attached to the instrument
                                    for this measurement
            possible values 0...MAX_NUM_ACCSRY  */
char acc_name[MAX_NUM_ACCSRY][14];
                                /*  names of accessories attached
                                    to this instrument each name
                                    is  14 bytes long including
                                    string terminator char */
char lens_name[S_NAME_SZ]; /* name of lens active in
                                    this measurement. */

int radnc_flag;                 /* radiance/irradiance etc flag
                    possible values:
                    RAD     radiance
                    IRRAD   irradiance
                    RADINT  radiant intensity */
struct adapt adp;               /*  copy of same from struct
                                    command */
    struct calc_w calw;         /*  holds calculated spectral
                                    parameters from meas */
struct cctemp cct;              /*  holds correlated coror temp
                                    values */
char expn[M_EXPN_SZ];   /* expansion area  */
```

```
if trig_type = DELAY then trig_param contains the delay in
seconds (1...32767) from when the last measurement was made
until the next meas is made

if trig_type = TOD then trig_param contains the hour (1..23)
when the next meas is to be made */

    int trig_min;              /* if trig_type = TOD trig_min
                                  contains the minute (0...59)
                                  when the next meas is to be
                                  made otherwise trig_min is
                                  meaningless */

    int trig_sec;              /* ditto for seconds (0...59)
                                  when next meas */

    int (*meas_trap)( int l_d, long itime );

            /* address of function to be called by adaptive
measuring (meas_adapt()) before each measurement If this
function pointer is zero no calls are made by meas_adapt().
This function is called before each measurement with two
arguments. The 1st argu is an int and = 0 if light meas, = 1 if
dark meas. The 2nd argu is a long and is the integration time in
milliseconds to be used in the measurement. meas_trap()
returns a 1 if the meas is to proceed or returns a 0 if the meas is
to be aborted. */

    int aper_code;  /* aperture to use for this meas

                    0 = use operator selected aperture.

                    (which aper actually used is reported
                    in struct meas)

                    APER1..APERMAX_APER = selected aper
                    to use */

    int num_acc;               /* number of external
                                  accessories attached to the
                                  instrument values
                                  1..MAX_NUM_ACCSRY */

     int acc_indx[MAX_NUM_ACCSRY]; /* list of indices of
external accessr's attached to the instrument for this
measurement run. These indices are used to index into struct
config.acc[] acc_indx[0] is index of lens used */
```

```
    int observer;                /*  field of view of standard
                                     observer for CIE tristimulus
                                     tables
                          possible values:
                        2 2 degree
                       10 10 degree */
    struct adapt adp;            /*  adaptive sens parameters
                                     for this meas */
    char aux_data[128];          /*  string of auxiliary data
                                     characters from the aux
                                     data clause. this string will
                                     be null is the aux data clause
                                     is absent      */
    int end_value;               /*  marker of end of struct */
  };
/*-------------------------------------------------------------*/
struct spectral
 {
 int size;                       /* num bytes in this struct  */
 int s_ver;                      /* version of this structure */
 float start_wav, end_wav, wav_incr;
 char expansion[S_EXPN_SZ];
 float aux_data[N_AUX_DATA];
 int s_d_size;                   /* num floats in data (below) */
 float data[MAX_V_PIX];          /* spectral data */
 };
/*-------------------------------------------------------------*/
```

```
struct accessory                /*  This structure holds
                                    descriptions flags and
                                    correction factors for all
                                    accessories (spectral and
                                    spatial), apertures, front
                                    filters and rear filters */
{
int size;                       /*  size of the structure NOTE
                                    - This may vary from on
                                    acc to another based on
                                    whether struct spec exists
                                    and its size */
char idate[22];                 /*  comp str for idate.exe */
char name[ACC_NAME_SZ];         /*  accessory name ASCII
                                    string  */
char date[26];                  /*  ASCII string date/time of last
                                    calibration of this accessory */
char oper[10];                  /*  calibration operator i.d. */
int acc_cat_num;                /*  number of this accessory in
                                    the Photo Research catalog
                                    of PR7xx accessories */
int r_lamp_cat_num;             /*  number of lamp used for
                                    relative intensity calib of
                                    this acc from Photo Research
                                    lamp catalog */
int a_lamp_cat_num;             /*  number of lamp used for
                                    absolute intensity calib of
                                    this acc from Photo Research
                                    lamp catalog */
int cal_flags[MAX_APER];        /*  calibration flags for this
                                    accessory with each aperture.
                                possible values :
                                CALIBRATED accsry is calibrated for this
                                mode of operation
                                UNCALIB accsry not calib for this mode */
```

```
float u_wav_lim;            /*  upper wavelength in
                                nanometers of use of this
                                acc */

float l_wav_lim;            /*  lower wavelength in
                                nanometers of use of this
                                acc */

int radnc_flag;             /*  radiance mode flag for this
                                accsry

                    possible values:

                      RAD       radiance

                      IRRAD     irradiance

                      RADINT    radiant intensity

                      NA        not applicable (useable for
                                either rad, irrad or rad int */

int lens_flag;              /*  indicates whether this
                                accessory is considered a
                                lens i.e. only one acc with
                                lens_flag = = TRUE can
                                be on the instrument at one
                                time */

double preset_corr[MAX_APER]; /* preset correction factor
                                 */

double rad_static_corr[MAX_APER];
                    /* radiometric correction factor */

double phot_static_corr[MAX_APER];
                    /* photometric correction factor */

double calib_lum;           /*  luminance in foot_lamberts
                                of lamp used in absolute
                                intensity calibr of this acc */

struct spectral spec[MAX_APER];
                    /* struct spectral that holds spectral
                    correction factors for this accessory. */

};

/*-----------------------------------------------------------*/
```

```
struct code_indx_tbl            /*  this struct is used in table
                                    form to associate such items
                                    as bandwidths with the index
                                    to the appropriate struct
                                    accessory. The end of the
                                    table is indicated by both
                                    code_value and acc_indx
                                    containing 0     */
 {
 int code_value;                /*  holds code value for
                                    associated acc_indx */
 int ucicl_code;                /*  holds the UCICL code for
                                    this accessory   */
 int acc_indx;                  /*  holds index into struct
                                    accessory[] for the above
                                    code_value      */
 };
/*-------------------------------------------------------------*/
/*-------------------------------------------------
 status.h -- 700 series 10-10-85 (phm)
 -------------------------------------------------*/
```

```
struct status
{
 int size;                      /* size of this struct */
 int s_ver;                     /* version of this structure */
 int measuring;                 /*  measuring
                                    not measuring
                                    weak signal
                                    low light level
                                    overload */
 int last_gain;                 /*  last gain used */
 int last_cycl;
 long last_integ_time;          /*  last integration time used */
 int cooler;                    /*  on
                                    off
                                    malfunction */
 int pressure;                  /*  Optical head chamber
                                    pressure
                        Possible values
                          LOWPRES
                          QUAL_OK */
 int head;                      /*  okay
                                    malfunction */
 int filter;
 int accessory;                 /*  accessory attached */
 int view_port;                 /*  open
                                    shut */
 int meas_shutter;              /*  open
                                    shut */
 int integ_time_typ;            /*  manual adaptive */
 int meas_aborted;              /*  user malfunction */
};
/*-------------------------------------------------------------*/
```

```
struct calib                    /*  calibration values read in
                                    from disc */
{
 int size;                      /*  size of this struct */
 char idate[22];                /*  comp str for idate.exe */
 char aper_name[S_NAME_SZ]; /* name of this aperture */
 char acc_name[S_NAME_SZ];  /* name of this accessory lens
                               */
 char date[26];                 /* date/time string of calibration */
 char oper[10];                 /* operator i.d. */
 int acc_cat_num;               /*  number of the accessory in
                                    the Photo Research catalog
                                    of PR7xx accessories used
                                    with this aperture */
 int w_lamp_cat_num;            /*  catalog num of lamp used
                                    to wave calib this aper */
 int r_lamp_cat_num;            /*  catalog num of lamp used
                                    for relative intensity  calib
                                    of this aperture */
 int a_lamp_cat_num;            /*  catalog num of lamp used for
                                    absolute intensity calib of
                                    this aperture */
 int radnc_flag;                /*  radiance mode flag for this
                                    aper/lens
          possible values:
            RAD         radiance
            IRRAD       irradiance
            RADINT      radiant intensity
            NA          not applicable    */
 int aper_num;                  /*  aperture number of aperture
                                    used in this calibration
                                    (1..MAX_APER) */
 int num_time_factors;          /*  number of time_corr and
                                    time_zones */
```

```
    double time_corr[NUM_T_FACT]; /* integration time
                                     correction factors */

    double time_zone[NUM_T_FACT];
                    /* lower integration time for each time_corr */

    double calib_lum[NUM_T_FACT];
                    /* luminance in foot-Lamberts of lamp used
                       at this integr time */

    double rad_static_corr;      /*  radiometric correction factor
                                     for this aper (absolute
                                     intensity scale calibration
                                     factor) (07/16/90 r.k.) */

    double first_v_pix;          /*  wavelength or position of 1st
                                     virt pixel (wavl in nm units)  */

    double last_v_pix;           /*  wavelength or position of last
                                     virt pixel (wavl in nm units)  */

    double bandw;                /*  bandwidth in nm of instru
                                     with this aper */

    double v_pix_incr;           /*  virt pixel wavelength
                                     increment in nm */

    int num_v_pix;               /* number of virt pixels */

    int first_raw_pix;           /*  index of first raw pixel used
                                     for corrected measurements */

    int last_raw_pix;            /* ditto for index of last raw pixel */

    int start_pix[MAX_V_PIX];    /*  index of the first raw pixel
                                     of three consecutive raw
                                     pixels whose values are used
                                     to compute the value of each
                                     virtual pixel */

    float corr_fact[3 * MAX_V_PIX]; /* wavel corr factors
                                     (these are used to compute
                                     virtual pixel relative values
                                     from real pixel values */

    float rel_corr[MAX_V_PIX];   /* relative intensity
                                     correction factors for each
                                     virtual pixel (07/16/90 rk) */
};
/*-------------------------------------------------------------*/
```

```
struct cie_tbl        /* holds cie X, Y and Z tristimulus values */
  {
  int size;           /* size of this structure */
  float cie_tblx[MAX_V_PIX];      /* holds X tristimulus values
                                     */
  float cie_tbly[MAX_V_PIX];      /* ditto for Y */
  float cie_tblz[MAX_V_PIX];      /* ditto for Z */
  };
/*-----------------------------------------------------------------*/
struct cal_work  /* flags tell whether various stages of calib
have been performed (values are TRUE or FALSE) */
   {
 char ap_w[MAX_APER];  /* wavelength calib meas flags */
 char ap_r[MAX_APER];  /*   relative intens meas calib flags */
 char ap_wr[MAX_APER]; /* wavel and rel calib calc flags */
 char ap_a[MAX_APER];  /* absolute intens calib flags */
 char ac_r[MAX_NUM_ACCSRY];  /* relative intens flags*/
 char ac_a[MAX_NUM_ACCSRY];  /* absoulte intens flags */
   };
/*-----------------------------------------------------------*/
struct config
 {
 int size;      /* num bytes in this structure    */
 int s_ver;     /* version of this structure */
 char idate[22]; /* comp str for idate.exe */
 int m_size;    /* num bytes in struct meas   */
 int m_d_size;  /* num of flt pnt values in struct meas.data[] */
 int s_size;    /* num bytes in struct spectral   */
```

```
int s_d_size;                   /*  num of flt pnt values in struct
                                    spectral.data[] */

char ver_num[12];               /*  ASCII string version num
                                    of this software */

char instr_title[10];           /*  ASCII string title of
                                    instrument */

char ser_num[10];               /*  ASCII string serial number
                                    of instrument */

char calib_date[10];            /*  ASCII string date of last
                                    calibration */

int head_type;                  /* type of meas head
                                    possible values:
                                    PR701
                                    PR703
                                    PR704
                                    PR719    */

int shtr_type;                  /*  type of meas/zero shutter in
                                    head

                                possible values:
                                  PULSE     an i/o strobe changes shutter state
                                  MOTOR     a stepper motor is used to control
                                            shutter */

int num_aper;                   /*  number of apertures in this
                                    measuring head */

int dum_sys;                    /*  holds dummy system control
                                    flag

                                if TRUE, no head measurements are possible

                                if FALSE, head measurements are possible */

int io_base;                    /*  holds base i/o address of
                                    meas head controlled by this
                                    struct */
```

int use_flags[NUM_MODES]; /* use flags - These flags determine whether this instrument can be used in the 6 basic modes of operation (WAV, POS, ARC, COL, DAT and CHR).

index values 0 = WAV_INDX

1 = POS_INDX

possible values :

ALLOWD okay to use this instru in mode

NOT_ALWD use disallowed in this mode */

int num_r_pix; /* number of real pixels for this head
possible values:
1...MAX_R_PIX */

int num_v_pix; /* number of virtual pixels for this head
possible values:
1...MAX_V_PIX */

double first_v_pix; /* wavelength or position of 1st virt pixel
(wavl in nm units) */

double last_v_pix; /* wavelength or position of last virt pixel
(wavl in nm units) */

double bandw; /* bandwidth in nm of virt pixels */

double v_pix_incr; /* virt pixel increment (if wavl in nm) */

int detector; /* detector in measuring head
possible values:

RET128 Reticon with 128 elements (pr719)

RET256 ditto with 256 elements (pr710)

RET256X256 2 dimensional Reticon 256 * 256 */

int overload; /* overload raw data value */

```
int noise;                      /*  noise raw data value */

long min_i_time;                /*  min integration time in ms */

long max_i_time;                /*  max integration time in ms */

int min_l_d;                    /*  minumum raw light - raw
                                    dark of peak value of a meas
                                    that causes "low light" status */

int pho_units;                  /*  photometric units indicator
                        possible values:

                    ENGLISH  footLamberts, footcandles

                    METRIC   Cd/m2, Lux      */

struct adapt adp;               /* adaptive sensitivity parameters */

struct cie_tbl cietbl[2];       /*  holds trisitmulus values for
                                    2 degr and 10 degr observers.
                                    index = 0 for 2 degr
                                    index = 1 for 10 degree
                                    observer  */

struct code_indx_tbl aper_tbl[MAX_APER];
                        /* table of aperture codes and their
                        corresponding indicies into struct calib *cal[]
                         */

struct code_indx_tbl ext_acc_tbl[MAX_NUM_ACCSRY];
                        /* table of external accessory numbers, the
                        equivalent UCICL code and their associated
                        indices into struct  accessory acc[]  */

int acc_cat_num[MAX_NUM_ACCSRY];
                        /* accessory catalog numbers of the acc's by
                         acc index */

struct calib *cal[MAX_APER];  /* pointers to struct calib's */

char cal_file[MAX_APER][ACC_FILE_NM_SZ];
                        /* names of files containing calibration factors
                         */

int num_acc;                    /*  number of accessories
                                    included with this instrument
                                 */
```

```
struct accessory *acc[MAX_NUM_ACCSRY];
                        /* pointers to struct acc's */
 char acc_file[MAX_NUM_ACCSRY][ACC_FILE_NM_SZ];
                        /* names of files conatining struct accessory */
struct cal_work cw;              /* work area for calibration */
int end_value;                   /* marker of end of struct */
};
/*-------------------------------------------------------------*/
struct spare                     /*   general purpose struct to
                                      hold auxiliary calc's */
 {
 int size;
 float x[20];
 int i[10];
 long ii[10];
 char c[80];
 };
/*-------------------------------------------------------------*/
struct pointers {
void *files;                     /* 00 - ptr to files array*/
void (*edumfunc)(void);          /* 01 - ptr to dummy function*/
void (*eacie)(void);             /* 02 - ptr to asc cie( )    */
void (*ebcie)(void);             /* 03 - ptr to bin cie( )    */
void (*eacorrct)(void);          /* 04 - ptr to asc corrected( )*/
void (*ebcorrct)(void);          /* 05 - ptr to bin corrected( )*/
void (*earawdrk)(void);          /* 06 - ptr to asc rawdark( )*/
void (*ebrawdrk)(void);          /* 07 - ptr to bin rawdark( )*/
void (*earawdif)(void);          /* 08 - ptr to asc rawdiff( )*/
void (*ebrawdif)(void);          /* 09 - ptr to bin rawdiff( )*/
```

```
void (*earawlit)(void);          /* 10 - ptr to asc rawlit( )*/
void (*ebrawlit)(void);          /* 11 - ptr to bin rawlit( )*/
void (*eameas)(void);            /* 12 - ptr to asc meas( )*/
void (*ebmeas)(void);            /* 13 - ptr to bin meas( )*/
void (*eacct)(void);             /* 14 - ptr to asc cct( )*/
void (*ebcct)(void);             /* 15 - ptr to bin ccft( )*/
void (*eastat)(void);            /* 18 - ptr to asc stat( )*/
void (*ebstat)(void);            /* 19 - ptr to bin stat( )*/
void (*eacode)(void);            /* 20 - ptr to asc eacode( )*/
void (*ebbkgnd)(void);           /* 21 - ptr to bin bkgnd( )*/
void (*easpare)(void);           /* 22 - ptr to asc spare( )*/
void (*ebspare)(void);           /* 23 - ptr to bin spare( )*/
void (*emeas)(void);             /* 24 - ptr to meas_func( )*/
struct  meas *pmm;               /* 25 - ptr to struct meas*/
struct  cmd_wndw *pwn;           /* 26 - ptr to struct cmd_wndw*/
struct  result_wndw *pr_wn;      /* 27 - ptr to struct result_wndw*/
struct  calc_w *pcalw;           /* 28 - ptr to struct calc_w*/
struct  calc_s *pcals;           * 29 - ptr to struct calc_s*/
struct  meas_data *mmd;          /* 30 - ptr to struct meas_data*/
struct  in_data *pld;            /* 31 - ptr to struct int_data*/
struct  int_data *pdd;           /* 32 - ptr to struct int_data*/
struct  int_data *psd;           /* 33 - ptr to struct int_data*/
struct  flt_data *pcd;           /* 34 - ptr to struct flt_data*/
struct  command *pcmd;           /* 35 - ptr to struct command*/
struct  config *pcnf;            /* 36 - ptr to struct config*/
struct  status *pstatp;          /* 37 - ptr to struct status (N/A
                                    for 704/714)*/
struct  spare *pspare;           /* 38 - ptr to struct spare*/
```

```
    void (*xcorrct)(void);          /* 39 - ptr to corrected calc
                                    routine*/

    void (*xciecalc)(void);         /* 40 - ptr to cie calc routine*/

    void (*xcctemp)(void);          /* 41 - ptr to cct calc routine*/

    void (*xhook0)(void);           /* 42 - ptr to spare hook routine*/

    void (*xhook1)(void);           /* 43 - ptr to spare hook routine*/

    int  *hook_rtn_flag;            /* 44 - ptr to hook_rtn global used
                                        by hookx( )*/

    int (*zhook0)(char *auxfil);    /* 45 - ptr to hook funct @ end of
                                        init*/

    int (*zhook1)(struct config *cnf, /* 46 -ptr to hook funct @
    struct command *cmd,                meas_func entry */
  struct meas *mm );
    int (*zhook2)(long time,        /* 47 -ptr to hook funct @
    int gain,                           meas_func exit */
    struct config *cnf,

  struct command *cmd,

  struct meas *mm );
    int (*zhook3)(long time,        /* 48 -ptr to hook funct @
    int gain,                           meas_dark entry */
   struct config *cnf,

  struct command *cmd,

  struct meas *mm );
    int (*zhook4)(struct config *cnf, /* 49 -ptr to hook funct@
    struct command *cmd,                correct entry */
   struct meas *mm );
    int (*zhook5)(struct config *cnf, /* 50 -ptr to hook funct@
    struct command *cmd,                correct exit */
  struct meas *mm );
    int (*zhook6)(struct config *cnf);  /* 51 -ptr to hook funct @
    meas_init entry */

    int (*zhook7)(struct config *cnf);  /* 52 -ptr to hook funct @
                                        meas_init exit */

    int (*zhook8)(char *p,              /* 53 -ptr to hook funct @
                                        struct config *ccnf,toolpars
                                        entry */

    struct command *ccmd,
    struct command *prot_ptr);
```

```
int (*zhook9)(char *p,                  /* 54 -ptr to hook funct @
struct config *ccnf,                    toolpars exit */

struct command *ccmd,

struct command *prot_ptr );

int (*zhook10)(char *pspare);           /* 55 -ptr to hook funct @
                                        spare.asc & .bin entry */

int (*zhook11)(char *pspare,            /* 56 -ptr to hook funct @
char *ptr1);                                spare.asc & .bin exit  */

int (*zhook12)(                         /* 57 -ptr to hook funct @
struct config *cnf,                         cctemp entry */

struct command *cmd,

struct meas *mm );

int (*zhook13)(                         /* 58 -ptr to hook funct @
struct config *cnf,                         cctemp exit */

struct command *cmd,

struct meas *mm );

int (*zhook14)(                         /* 59 -ptr to hook funct @
struct config *cnf,                         ciecalc entry */

struct command *cmd,

struct meas *mm );

int (*zhook15)(*cnf,                    /* 60 -ptr to hook funct @
struct config *cnf,                         ciecalc exit */

struct command *cmd,

struct meas *mm );

  };

#define NUM_PTRS   sizeof(struct pointers) / sizeof ( char *)

/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
```

Listing 2 - PR704STR.H

This file contains information relating to hardware and defines **struct hd_stat.**

```
/*

    file pr704str.h

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

 Copyright (C) 1988 Photo Research Division of
Kollmorgen Corp.

 All rights reserved.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

structures for the PR704 cpu and the host cpu (IBM PC)

03-29-88  r.k.  created

*/

/*-------------------------------------------------------------*/
    struct hd_stat1
    {
    int hd_id;                  /*  ID (7bits) from head Bits 5
                                    and 6 are array size code.
                                    Bits 0..4 are head id. (if head
                                    not present = -1) */

    unsigned int ctlr_id;       /*  ID of controller - should
                                    match low 5 bits of hd_id */

    unsigned int arry_sz;       /*  number of pixels in detector */
    unsigned long sw_sn;        /*  PR704 CPU software serial
                                    number */
```

```
unsigned long sw_ver;          /* PR704 CPU software version
                                  number */

char idate[22];                /* holds date of build of
                                  firmware put here by utility
                                  idate.exe in place of a
                                  compare string
                                  "???***$Ph...." */

unsigned int num_aper;         /* number of apertures on this
                                  instr */

unsigned int shtr_type;        /* View and Zero shutters
                                  type */

unsigned int rom_cksum;        /* check sum of all but last 4
                                  bytes of the PR704 CPU
                                  PROM. (The last 4 bytes
                                  are reserved locations in the
                                  80188 processor).  */

};
struct hd_stat2
{
unsigned char dip_sw;          /* current setting of option DIP
                                  switch on the PR704 controller
                                  (carrier board) bit 0
                                  0 = select Zero shutter
                                  1 = select View shutter bits
                                  1..3 specify a simple diagnostic
                                  to run. values of bits 1..3:
                                  0 = normal (no diag's)
                                  1 = move shtr continuously
                                  2 = move aperture cont.
                                  3 = calibrate a shutter
                                  4 = not specified yet
                                  5 = not specified yet
                                  6 = RAM test
                                  7 = normal (no diag's)
```

```
                        bits 4..5   not assigned yet
                        bit 6       0 = stepper shtr ( PR704)
                                    1 = single pulse shtr
                        bit 7       0 = 100 usec pix clock
                                    1 = 20 usec pix clock  */
unsigned char z_shtr;   /*          current position of zero/measure
                                    shutter.
                                        0 = open
                                        1 = closed  */
unsigned char v_shtr;   /*          current position of view/measure
shutter.
                                        0 = open
                                        1 = closed  */
unsigned char aper;     /*          currently selected aperture
                                    values 1..2  */
unsigned char cooler;   /*          current status of TE cooler
                                        0 = NOT at operating
                                            temmperature
                                        1 = at operating
                                            temperature  */
unsigned char pressure; /*          detector housing pressure indicator
                                        0 = pressure = 2 psi
                                        1 = pressure 2 psi  */
unsigned char m_v_sw;   /*          current setting of manual view
                                    switch.
                                        0 = NOT pressed (normal)
                                        1 = pressed  */
unsigned char m_z_sw;   /*          current setting of manual zero
                                    shutter switch.
                                        0 = NOT pressed (normal)
                                        1 = pressed  */
```

```
unsigned int num_faults;        /*  number of faults pending */
};
struct hd_stat3
{
unsigned char shtr_mode;        /*  current shutter mode
                                    0 = AUTO
                                    1 = MANUAL    */
unsigned char sync_mode;        /*  current sync mode  */
unsigned int px_10xclk;         /*  current period in usec of
                                    the 10X pixel clock  */
unsigned int min_i_time;        /*  minimum integration time
                                    in ms based on pix clk rate
                                    and array size rounded up
                                    to next integer */
};
struct hd_stat
        {
         struct hd_stat1 s1;
         struct hd_stat2 s2;
         struct hd_stat3 s3;
        };
/*-------------------------------------------------------------*/
struct step_pos_calib
              {
int edge;                       /*  edge of step_pos_sens to
                                    calibrate positiion  on
                                    0  = high to low edge
                                    1 = low to high edge   */
int calib_pos;                  /*  meaning of calibrated
                                    position values  OPEN or
                                    CLOSED */
```

```
int bias;                       /*  number of steps past calib
                                    positionto be at exactly at
                                    OPEN or CLOSED state   */

};
struct stepper
{
struct step_pos_calib pos[2];   /*  indexed by Z_INDX or
                                    V_INDX  */

unsigned s_pulse_width;         /*  width of stepper pulse (low)
                                    in units of 0.5 usec.  */

unsigned int calib_lat;         /*  latency for calibrating in
                                    units of 0.5 usec.  */

unsigned int run_in;            /*  number of pulses to allow at
                                    start of stepper position calib
                                    before looking for the pos
                                    sense signal */

unsigned int on_delay;          /*  time in 0.5 usec units after
                                    turn on stepper motor until
                                    give it
                                    its pulse  */

unsigned int off_settle;        /*  time in milliseconds to leave
                                    motor off  */

int num_pulses;                 /*  number of pulses to change
                                    */

unsigned int s_puls_lat[50];    /*  table of times between
                                    stepper pulses in units of
                                    0.5 usec. */

};
/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
```

Appendix FS
Fast-Scan for the PR-704/714

The Fast-Scan software allows performance of high-speed, high resolution, full-spectrum multiple scans with the PR-704/714 SpectraScan Systems and Toolkit.

During a measurement sequence, a single dark current measurement is performed for the user input fixed integration time. Next the requested number of sample measurements are performed.

The dark current data is then subtracted from all sample measurements. Between sample scans, the detector is purged for the minimum system integration time (13 or 26 milliseconds).

Since the Fast-Scan software requires the Toolkit option, the user must have Toolkit installed according to the instructions provided in the Toolkit Manual prior to executing the Fast-Scan program.

After installing Toolkit by typing the Toolkit executable, **tls_res**, the user loads FAST-SCAN using the following procedure.

Step 1 Verify that the FAST.EXE and program files from the Toolkit program diskette are located in the same directory as the Toolkit program.

If these files were not already copied during the Toolkit installation copy them now.
(e.g. C >**Copy A: *.* C:\TOOLS**)

Step 2 Go to the Toolkit directory (e.g. C:>**cd\TOOLS**).

Step 3 At the prompt (e.g. C:\TOOLS >), type **f a s t** and press **ENTER**.

After a few seconds, the following HEADINGS Menu screen appears as shown in Figure 1 below.

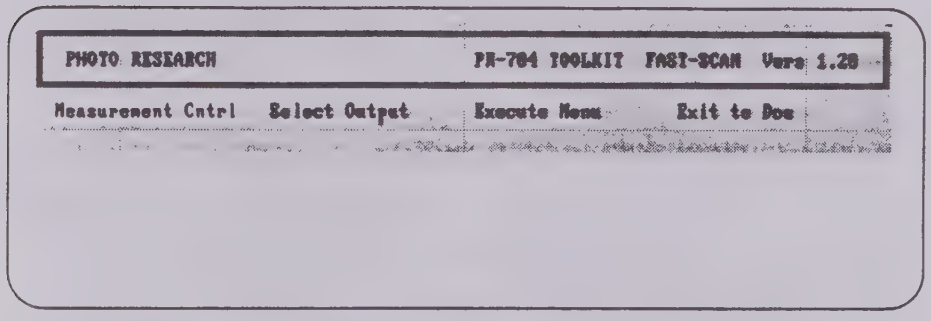


Figure 1. HEADINGS Menu

Headings Menu

The initial FAST SCAN screen contains four menu headings used to control the Fast-Scan software: **Measurement Cntrl, Select Output, Execute Menu,** and **Exit to Dos.**

1. **Measurement Cntrl:** Used to establish the integration time and the number of scans per measurement sequence.
2. **Select Output:** Used to establish the output data type and Destination Directory.

3. **Execute Menu:** Start measurements sequence.

4. **Exit to Dos:** Quits the program and returns the user to DOS.

The user selects the desired menu heading by using the right and left keyboard cursor arrow keys to move a reverse video highlight bar along the HEADINGS Menu. After highlighting a selection, the user presses the **ENTER** key to access the selected HEADING's pull-down menu.

Selecting Parameters in a HEADING

After pressing the **ENTER** key from a HEADINGS Menu selection, the user will notice a blinking cursor at the initial selection in the parameter list. Depending on the type of parameter, the user can either type in a number (e.g. #of integrations), or type in a **y** or **n** (to select or deselect a parameter).

After the user completes the first parameter in the list, the blinking cursor automatically moves to the next parameter.

Upon leaving the last parameter at the bottom of the menu, the following prompt appears at the bottom of the screen:

Do you want to exit this form [y/n]

Press **y** to exit the menu and accept changes; **n** to re-enter the menu and change additional parameters.

Measurement Cntrl Heading

Two parameters are available under the **Measurement Cntrl** heading: **Integration Time** and the **Number of measurements**, as shown in Figure 2 below:

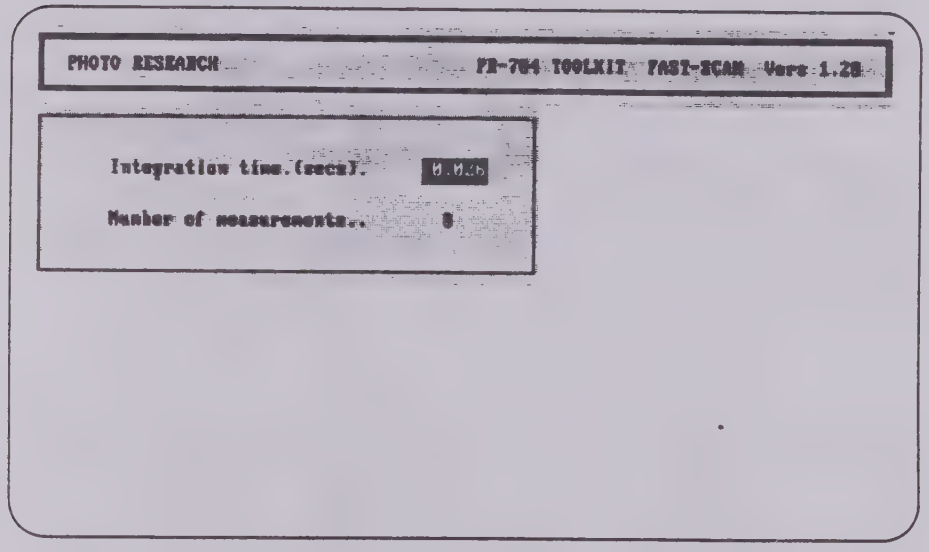


Figure 2. MEASUREMENT CNTRL Parameters

1. **Integration Time:** Integration time is the length of time the shutter remains open to collect signal from the sample. The integration time is precise to thousandths of a second (milliseconds). If the displayed integration time is correct, press **Enter, or** type in the desired time. **FAST_SCAN** defaults to the shortest system integration time. The maximum integration time for the PR-704/714 is 60.000 seconds.

The integration time appears once the fifth digit is entered and the blinking cursor is automatically repositioned at the **Number of measurements** parameter.

NOTE

The decimal point is fixed, allowing a two-digit whole number and a three digit decimal.

2. **Number of measurements:** The Number of measurements parameter lets the user enter the desired number of scans in a measurement sequence.

   Type the desired number of measurements (up to 500) as a three-digit number and press **Enter.**

NOTE

The system checks the amount of space available in your system's RAM (Random Access Memory). If there is not enough memory space for the number of measurements selected, a warning message appears at the bottom of the screen indicating the maximum number of scans available.

Select Output Heading

The **Select Output** heading lets the user select or deselect parameter information for inclusion in the measurement output for a series of scans.

This menu includes selection of up to four files (**Corrected, CIE, CCT, Raw Light**), two data formats (**ASCII and Binary**) and entry of a **Destination Directory**.

1. **Corrected data:** Measured data corrected for absolute spectral radiance in units of $w/sr/m^2$.

2. **CIE:** Tristimulus values of X, Y, and Z, and chromaticity coordinates x and y.

3. **CCT:** Correlated Color Temperature in Kelvins.

4. **Raw Light:** Uncorrected light signal (light and dark counts) collected from the photo detector. The values reported are A/D converter counts for each detector element.

5. ***.ASCII file:** Readable and editable text file containing measurement data.

6. ***.BIN:** File containing measurement data that can be used for calculations, but is not readable or editable. (All options must be set to (Y) for use by SpectraView).

   NOTE:

   For multiple scans with the binary file set to (Y)es, only the last measurement in the sequence is saved to disk.

7. **Destination directory:** Where measurement files are to be stored (e.g. **C:\TOOLS**).

Current parameter settings are displayed, and the user types either **y** or **n** to select or deselect from the available options. See Figure 3.

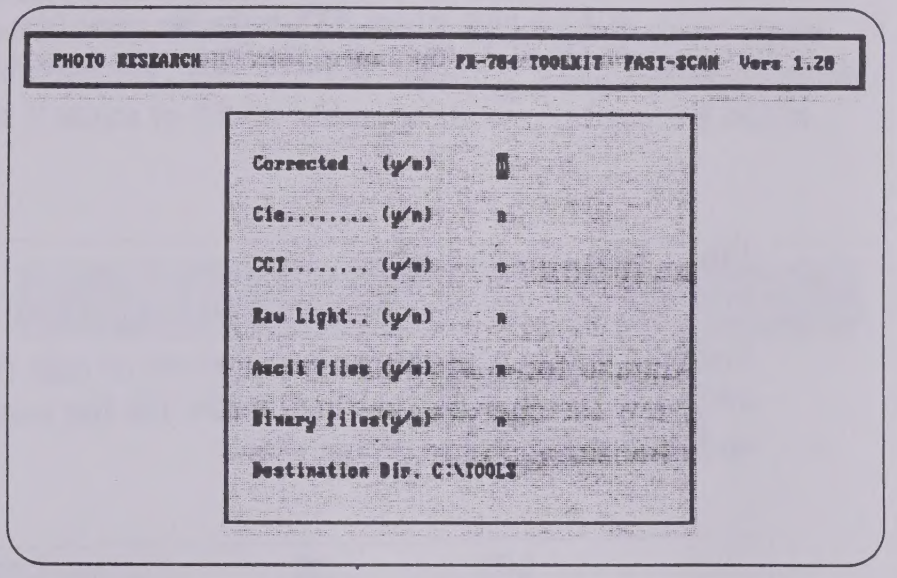


Figure 3. SELECT OUTPUT Parameters

After typing in the directory (standard DOS rules apply) where you want the data stored, press **Enter**, then type **y** at the *Do you want to exit this form [y/n]* prompt to return to the HEADINGS Menu.

If no entry is made, the data will be stored to the directory containing the FAST program.

NOTE:

It is recommended that files be stored to the hard disk to avoid running out of disk space during a measurement sequence.

The Destination Directory must be changed with each measurement series, or the new data will overwrite the previous data files.

Data files are numbered starting at XXX .001 and stop at XXX.nnn, where XXX is the file name (e.g., ACORRCT) and nnn is the final measurement in the series (e.g., 500).

Data File Descriptions

See the Toolkit Manual for descriptions of data file formats and content. Review the chart below for file names, file sizes and approximate file storage time.

	CCT DATA	CIE DATA	CORRECTED	RAW LIGHT DATA
FILENAME (ASCII)	ACCT.XXX	ACIE.XXX	ACORRCT.XXX	ARAWLIT.XXX
TOOLKIT FILENAME (ASCII)	CCT.ASC	CIE.ASC	CORRCT.ASC	RAWLIT.ASC
BYTES	54	258	2742	2225
STORAGE TIME* (ASCII)	6 MIN	5 MIN	7 MIN	3 MIN

*Storage times are based on 100 scans on a hard disk system at half-capacity running at a clock rate of approximately 6MHz without a math co-processor. Storage time will vary according to the user system configuration.

Execute Menu Heading

Before performing any measurement options included in this heading, all other parameters must be established.

The **Execute Menu** heading contains four options: **Start Immediately, Ext Trigger 1st only, Ext Trigger all meas,** and **Return to Main Menu** as shown in Figure 4 below:

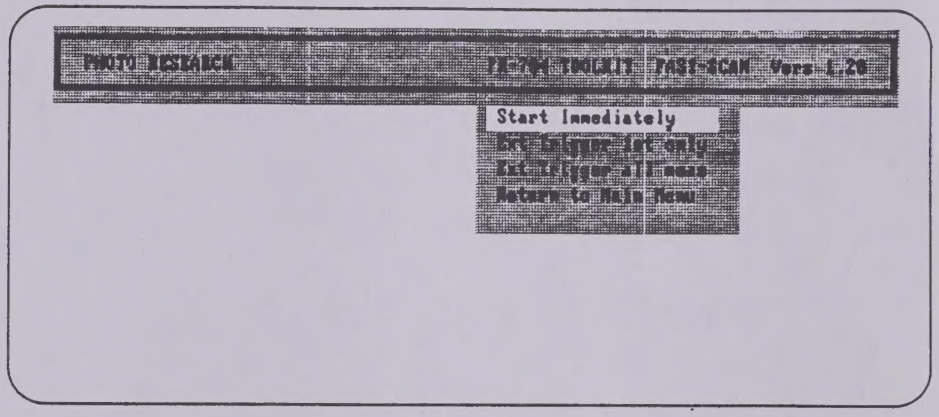


Figure 4. Execute Menu

1. **Start Immediately:** Press **Enter** to initiate a dark current measurement, which is followed by the number of measurements selected.

2. **Ext(ernal) Trigger * 1st only:** Press **Enter** to allow an external trigger to start the first measurement - which is followed immediately by the remaining selected number of measurements.

3. **Ext(ernal) Trigger * all meas:** Press **Enter** to specify the requirement of an external trigger for each measurement in the sequence.

4. **Return to Main Menu:** Press **Enter** to exit the Execute Menu and return to the HEADINGS Menu.

Select the desired option by moving the cursor arrow keys to highlight it, then press **Enter** to initiate it.

NOTE

***See the Toolkit Appendix for explanations of available Trigger Mechanisms (external SYNCH and keyboard mechanisms are allowed.)**

Measurement Procedure

When a measurement series is initiated, the software automatically performs the following steps:

Step 1 A dark current measurement is taken at the selected integration time.

Step 2 The measurements are taken and each raw data set is stored to RAM.

The detector is purged between measurements for the minimum integration time of the system (13 or 26 ms).